A Newton's Method Finite Element Algorithm for Fluid-Structure Interaction

by

Gabriel Balaban

Thesis
for the degree of
Master in Mathematics

(Master i Matematikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

October 2012

Det matematisk-naturvitenskapelige fakultet
Universitetet i Oslo

Acknowledgments

This Master's thesis was worked on during the course of an exciting year at the Centre for Biomedical Computing at Simula Research Laboratory. Many thanks to the staff and fellow students for creating an amazing working environment which was both enjoyable and productive.

I would especially like to thank my two thesis supervisors Anders Logg and Marie Rognes for setting me on the path and guiding me patiently on it. Along the way there were many questions and technical issues, and I greatly appreciate the generosity with which Anders and Marie showed me with regards to their availability and time. Thank you Anders for giving me inspiration and for your insightful explanations. Thank you Marie for often getting to the heart of a seemingly complex issue and gently showing me the way forward.

Several good ideas in this thesis, including the Jacobian reuse technique, were obtained while working on the Finmag project at the University of Southampton. Many thanks to Professor Hans Fangohr, the Finmag project leader, for getting me involved and giving me my first experience with agile software development.

Finally I would like to thank my partner Lena Gross and daughter Lyra Balaban for their love and support, and for giving me something to look forward to when the day's work is done.

Gabriel Balaban, Oslo, September 2012

Contents

Chapter 1

INTRODUCTION

Fluid-structure interaction (FSI), is said to occur when a fluid interacts with a solid structure. Typically the flow of the fluid exerts a force on the surface of the structure, causing the structure to deform, and altering the flow of the fluid itself. These types of problems are so called multi-physics problems as the physics of fluids and solid materials are fundamentally different. Additionally, FSI modeling requires the consideration of a set of interface conditions, which model how the fluid and solid interact.

FSI problems are of crucial importance in engineering. The design of airplane wings, bridges and engines are all examples where FSI is considered. Often fluid-structure interactions can be oscillatory, which can have drastic consequences for the structure, which is stressed repeatedly over time in exactly the same places. One example of this is the Tacoma Narrows bridge, which collapsed due to aeroelastic fluttering in 1940, the same year that it was built.

Another common application of FSI is biomedicine. Blood flow in arteries and air flow in the human repository system are typical examples where FSI must be considered as the deformation of the surrounding tissue cannot be ignored. Often the densities of the fluid and structure are very close in biomedical applications, causing the movements of the structure and fluid to become very sensitive to one another.

In many cases the computer simulation of FSI systems is preferable to physical experimentation. In the biomedical case physical experiments are often unethical. In the engineering case they can be very expensive, for example if a large amount of material is involved, or a large number of test cases are considered.

In this thesis, we investigate and design a numerical solver for fluid-structure interaction based on a monolithic finite element formulation in which the entire FSI problem is solved as one problem using Newton's method. The computer implementation is carried out using the software developed as part of the FEniCS project [Logg et al., 2012a].

1.1 Thesis Objectives

The objectives of this thesis are to:

- formulate a Newton's method algorithm for the solution of the mathematical model of fluid-structure interaction which is presented in [Selim, 2011];
- implement a finite element FSI solver using the above mentioned algorithm and make it publicly available via the software package CBC.Solve (https://launchpad.net/cbc.solve/);
- examine the performance of the Newton's method solver for various test problems and compare the results to the existing fixed point solver of CBC.Solve;
- develop and test optimizations to the FSI Newton's method algorithm which can be used in a variety of implementations.

1.2 Overview of FSI Solution Methods

Computer software for the solution of FSI problems can be roughly divided into *monolithic* and *partitioned* approaches. In the monolithic approach the computer code is written as one solver which handles an entire FSI problem. In contrast to this is the partitioned approach, which involves combining seperate solvers for the fluid and structure subproblems along with a some sort of coupling algorithm. The partitioned approach has the advantage of modularity, meaning that previously designed solvers can be tested individually, and combined and exchanged with ease. The monolithic approach however enables more compact coding as all of the program logic can be managed in one place.

At the highest level, FSI problems are typically solved using either *fixed point iteration* or *Newton's method*. With a partitioned approach, the fixed point iteration scheme is very natural to use, as the architecture of the algorithm mimics that of the computer code. Partitioned fix point solution involves the successive calling of the subsolvers, and stopping when a convergence criterion is met. This solution method is used in CBC.Swing's fixed point solver [Selim, 2012].

In FSINewton, the software package developed in this thesis, a monolithic architecture is used along with Newton's method. This involves the repeated assembly of a global system of linear equations, which is used to solve all of the variables in the system simultaneously. This approach has been successfully applied in Hron and Turek [2006] for a biomedical application, and in Heil [2004] for an engineering application.

Newton's method is also possible in a partitioned approach. If a pure Newton's method is used this involves having various sub solvers assembling their respective blocks in the global linear system. This system can be solved simultaneously, but often a custom algorithm is used that solves the system in various stages [Dettmer and Perić, 2008, Fernández and Moubachir, 2005]. This allows the FSI interface to be physically moved as part of the solution process, resulting in an explicit treatment of the coupling and a slightly less robust method than that obtained by a simultaneous solve.

The strength of the coupling is a frequently discussed issue in the FSI literature. The stronger the coupling, the more challenging the problems that can be solved. Fixed point iteration schemes and quasi Newton's methods generally have weaker couplings, whereas full Newton's methods involving an exact Jacobian are said to have stronger couplings.

1.2.1 Jacobian Calculation and Quasi Newton's methods

One of the challenges in implementing Newton's method is the calculation of the derivatives of the fluid variables with respect to the changing fluid domain geometry. This is done in Fernández and Moubachir [2005] via shape differentiation. In this thesis an alternative approach is used that involves introducing a variable to track the deformation of the fluid domain. This variable is then incorporated into the fluid equations via a domain mapping. This procedure makes the dependency of the fluid variables on the changing geometry explicit, and allows the straightforward calculation of the derivatives, which can even be automatically handled by a computer using symbolic differentiation.

In many works a quasi Newton's method is used in which parts of the global Jacobian matrix are simplified or ignored. This leads to less calculation, both human and machine, less coding, and reduced matrix assembly times. However the cost of this is paid in reduced convergence properties which lead to more overall Newton iterations, and a less robust method. Approximate Jacobians can be obtained via finite difference schemes [Matthies and Steindorf, 2003, 2002, Tezduyar, 2001, Heil, 2004], or by directly dropping terms [Tezduyar, 2001, Gerbeau and Vidrascu, 2003, S.Deparis, 2004, S.Deparis et al., 2004, Gerbeau and Vidrascu, 2003].

1.2.2 The ALE Formulation

The presence of large structure deformations cause a particular modeling difficulty in FSI simulations. Large structure deformations imply large changes in the geometry of the fluid-structure interface, and these cannot be tracked properly in the classical Eulerian framework of fluid mechanics. In order to overcome this difficulty a hybrid framework is used, the Arbitrary Lagrangian-Eulerian (ALE) framework. This framework combines

the precise tracking of material boundaries of the Lagrangian framework, along with the decoupling of computational mesh and material of the Eulerian framework, allowing the accurate modeling of the fluid flow and its deforming structure boundary.

As a computational method, ALE is quite established. It's history can be traced back to 1964, when it was introduced in Noh et al. [1964], under the name 'coupled Eulerian–Lagrangian' in a finite difference framework to solve two-dimensional hydrodynamics problems with moving fluid boundaries. The method was later extended to fluid dynamics problems in two and three dimensions in Hirt et al. [1974], Stein et al. [1977] respectively. The application to fluid-structure interaction came shortly after, in Belytschko and Kennedy [1978], which also introduced a finite element formulation into ALE modeling. Later on in Donea et al. [1982], transient FSI problems with ALE are considered.

In contemporary works the ALE method remains popular in both fluid and solid mechanics for the handling of large deformations and free boundaries. In Nazem et al. [2009], ALE is used in geotechnical modeling, and in Li et al. [2005] it is applied to the analysis of jumping off of water.

In the context of FSI with ALE and the finite element method one often uses separate sets of elements for the fluid and structure domains, which are connected by shared vertices along the fluid-structure interface. This setup can handle large deformations well, but is quite ill- suited to handling large translations or rotations in the structure. These types of movements can quickly exhaust the ability of mesh smoothing algorithms to maintain mesh quality, and require frequent remeshing in the ALE framework. For this reason all the structures considered in this work are fixed at some point, and do not undergo any substantial rigid body motion.

For simulations involving a completely free structure, techniques other than ALE are more appropriate. A few of these techniques are the Immersed boundary method; [Peskin, 2002, 1977], the distributed Lagrange multiplier method/fictitious domain method; [F.P.T, 2004, R.Glowinski et al., 1999], Chimera schemes; [Stegera et al.], and eXtended Finite Element Method (XFEM), [Legay et al., 2006]. For an in depth discussion of these methods, see Wall et al. [2006].

Another possible alternative to ALE is Nitsche's method, which has been very recently applied to the Stokes problem using a fictitious domain and overlapping mesh approach in Massing et al. [2012b] and Massing et al. [2012a] respectively. Finally, a so called fully Eulerian description may be used, such as in Dunne and Rannacher [2006]. This involves the posing of the structure problem in the Eulerian framework and avoids the need for mesh movement and smoothing.

Chapter 2

STRONG AND WEAK FORMULATION OF THE FSI EQUATIONS

After having discussed fluid-structure interaction in general in the introduction we would now like to present the class of fluid-structure interaction problems that are considered in this thesis. We do this by first discussing continuum mechanics and frameworks and how they related to our FSI problem class, as well as introducing some notation. We then present three subproblems (fluid, structure and fluid domain) which when coupled together comprise a system of partial differential equations in strong form, which model FSI. Finally we derive the weak forms related to these equations.

2.1 Continuum Mechanics and Frameworks

The mathematical models considered in this work are *continuum mechanics* models, i.e. the matter under consideration is modeled as an infinitely divisible continuous mass, effectively ignoring all atomic theory. This is a reasonable assumption for the mass scales that are considered in common FSI simulations, as the presence of a large amount of atoms makes the interactions between any two negligible. Since physical systems are simulated in this thesis using the *finite* element method, it is necessary to limit the size of the material being modeled. This requires the definition of a *computational domain*, called Ω, where the simulation takes place.

Continuum mechanics models are formulated in a so called *framework*, which defines how the motion of the physical body is represented, and the relationship of the body to the computational domain. The framework allows the formulation of conservation laws, expressed as differential equations that govern the motion of the material. In order to distinguish one material from another, *constitutive laws* are required. These laws describe how a material reacts to forces acting upon it, called *stresses*. Stresses can come from outside of the material in the form of surface or body forces, or from the material itself, as in the case of an elastic object under a deformation.

Objects under stress will typically move in some way, and the spatial differences in the movement are called the *strains*. More details about continuum mechanics can be found in Gurtin [1981].

2.1.1 Domains and Mappings

The solution domain $\omega(t)$ for the FSI problem consists of two parts, the current fluid domain $\omega_{_F}(t)$ and current solid domain $\omega_{_S}(t)$, both of which can vary in time. The FSI interface, the common boundary of the two domains, is denoted $\gamma_{_{FSI}}(t)$. In order to map the FSI problem between a reference domain, $\Omega = \omega(0)$, and the current domain, a mapping between the two domains is introduced:

$$\Phi(t) : \Omega \longrightarrow \omega(t) \tag{2.1}$$

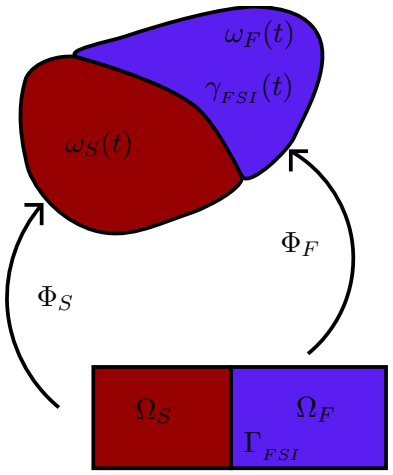

Figure 2.1: Illustration of the mappings $\Phi_{_S}$ and $\Phi_{_F}$ from the reference domains $\Omega_{_S}$ and $\Omega_{_F}$ to the current domains $\omega_{_F}(t)$ and $\omega_{_S}(t)$. The current and reference fluid-structure interfaces are marked $\gamma_{_{FSI}}(t)$ and $\Gamma_{_{FSI}}$ respectively.

The mapping Φ is further divided into the fluid domain and structure mappings $\Phi_{_F}$ and $\Phi_{_S}$ as illustrated in Figure 2.1. We define the gradient matrices of these mappings by $F_{_F} = \text{grad}\,\Phi_{_F}$ and $F_{_S} = \text{grad}\,\Phi_{_S}$, and the corresponding determinants by $J_{_F} = \det(F_{_F})$ and $J_{_S} = \det(F_{_S})$. Furthermore we let N_S denote the unit outer normal with respect to $\Omega_{_S}$ and N_F the unit outer normal with respect to $\Omega_{_F}$. In general, throughout this work the subscripts F and S are used to mark entities that are related to the fluid and structure respectively.

2.1.2 Frameworks

In order to model the various physical quantities of interest in an FSI system two frameworks are used in this work, the Lagrangian and the ALE (Arbitrary Lagrangian Eulerian) frameworks. A third framework, Eulerian, is described as a stepping stone leading up to the ALE framework. Fundamental to the frameworks are the concepts of *spatial* and *material* points. Material points are simply points in a material that move in the same way as the material does. Spatial points, denoted by hats in the following sections, are defined in a computational domain which acts as a 'window' through which the material is viewed.

Lagrangian Framework

In the Lagrangian framework material and spatial points coincide. This allows a simple description of material movement in which the changing geometry of the computational domain corresponds to the changing geometry of the material. This makes the Lagrangian framework a natural choice for the modeling of the deformation of solids. The primary variable of interest in this case is the displacement of each point from a given reference configuration, which means that all of the material that we are modeling can be followed from start to finish. The displacement of a point is a function of the original material coordinate X and the time t such that

$$D(X,t) = x - X = \Phi(X,t) - X \tag{2.2}$$

Here x denotes the current location of the material point that was at X at time 0. Figure 2.2 demonstrates the use of the Lagrangian framework in the simulation of a circle being deformed into an ellipse.

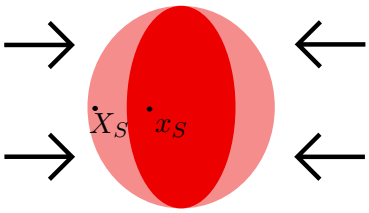

Figure 2.2: Deformation of a circular region in the Lagrangian framework. The point X_S in the reference configuration (shown as light red) is pushed to it's new position x_S in the current configuration.

In the Lagrangian framework the principle of the conservation of mass is simple to express. If we let ρ be the material density, then mass conservation

is given by

$$\frac{d}{dt}\int_{\Omega_S} \rho \, \mathrm{d}X = 0. \tag{2.3}$$

Similarly, the conservation of momentum, called the Cauchy Momentum equation, is given by

$$\frac{d^2}{dt^2}\int_{\Omega_S} \rho D \, \mathrm{d}X = \int_{\partial\Omega_S} \Sigma \cdot N \, \mathrm{d}S + \int_{\Omega_S} B \, \mathrm{d}X. \tag{2.4}$$

Here N is the unit outer normal, B the body force, and Σ the so called Cauchy stress tensor, which models the relationship between stress and strain in a solid continuum body. Here Σ is symmetric in order to satisfy the balance of angular momentum.

Eulerian Framework

In the Eulerian framework, material and spatial points are separated, meaning that the computational domain is considered separate from the material itself. Furthermore, the spatial points are kept fixed, and the primary quantity of interest is the velocity. This framework is the natural choice for the modeling of fluids, since the region of interest will typically have a lot of material flowing in and out across the boundaries, making the Lagrangian framework impractical. An illustration of the flow of a river modeled in the Eulerian framework is presented in figure 2.3.

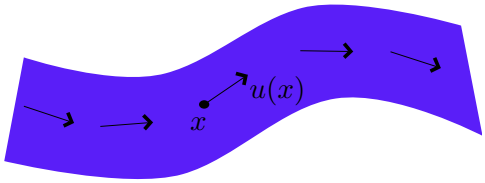

Figure 2.3: Flow of a river modeled in the Eulerian framework. At the spatial point x we are interested in the velocity of the fluid $u(x)$.

The conservation of mass in the Eulerian framework is given simply by the statement that the net flux of the material across the boundary should be zero, i.e. we expect no material to be created or destroyed inside the computational domain

$$\int_{\omega_F} \dot{\rho} + \operatorname{div} \rho u \, \mathrm{d}x = 0. \tag{2.5}$$

Here u denotes the material velocity, and Gauss's theorem has been used to write the equation as an integral over the inside of the domain.

The acceleration term in the Eulerian momentum equation is made a little more complex due to the separation of the material and spatial points.

If we want to know the momentum of the material particle x, lying on spatial point $\hat{x}$ at time t, we have to consider the path that the material point traveled, which we define by $\phi(X,t) = x(t)$. Using the chain rule, this gives us

$$\frac{d}{dt}[\rho u](x,t) = \frac{d}{dt}[\rho u](\phi(X,t),t) = [\dot{\rho u}] + \text{grad}\,[\rho u]\cdot \dot{x} = [\dot{\rho u}] + \text{grad}\,[\rho u]\cdot u, \tag{2.6}$$

which is called the material derivative. In this case ρ may or may not depend on time and space.

The Cauchy momentum equation in the Eulerian framework is then given by

$$\int_{\omega_F} (\dot{\rho u}) + \text{grad}\,(\rho u)\cdot u \,\mathrm{d}x = \int_{\partial\omega_F} \sigma\cdot n \,\mathrm{d}s + \int_{\omega_F} b \,\mathrm{d}x, \tag{2.7}$$

where σ is the usual Cauchy stress tensor, n the unit outer normal, and b a per volume body force.

Arbitrary Lagrangian Eulerian Framework

The ALE framework is very similar to the Eulerian framework, only the spatial points are no longer kept fixed. It is Eulerian in the sense that our primary variable of interest is still the material velocity $u(x,t)$, and Lagrangian in the sense that the spatial points are modeled in a Lagrangian framework. The position of the spatial points is denoted here $\hat{\phi}(\hat{X},t)$, and should fulfill $\hat{\phi}(\Omega,t) = \omega(t)$.

The ALE mass conservation equation is similar to the Eulerian one, only the computational domain is now time dependent

$$\int_{\omega(t)} \dot{\rho} + \text{div}\,\rho u \,\mathrm{d}x = 0. \tag{2.8}$$

The material derivative in the ALE framework is slightly different however

$$\frac{d}{dt}(\rho u)(x,t) = (\dot{\rho u}) + \text{grad}\,\rho u\cdot(u - \hat{v}). \tag{2.9}$$

It differs from the Eulerian material derivative only by the presence of the so called 'ALE term', $\hat{v} = \dot{\hat{\phi}}$. The explanation that follows is based on the one found in Dettmer and Perić [2006]. The density is here assumed constant and can be ignored for simplicity's sake.

Let $\mathbf{b}(t)$ denote a fluid body which overlaps the current computational domain $\omega(t)$ at some time t. Furthermore let $\mathbf{B}$ denote the initial configuration of $\mathbf{b}(t)$, and Ω the initial configuration of $\omega(t)$. In Figure 2.4 this setup is shown. An initial material point is denoted by X, an initial spatial point by $\hat{X}$, a current material point by x and a current spatial point by $\hat{x}$.

ϕ denotes the mapping from $\mathbf{B}$ to $\mathbf{b}(t)$, and $\hat{\phi}$ the mapping from Ω to $\omega(t)$. Since the current fluid body and the current computational domain overlap, there exists a mapping ψ such that $\hat{X} = \psi(X,t) = \hat{\phi}^{-1}(\phi(X,t))$.

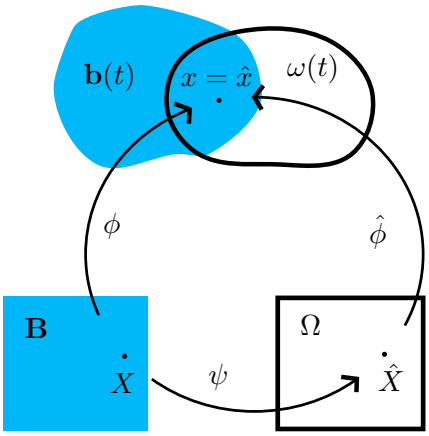

Figure 2.4: Domains and mappings in the ALE framework.

The goal is to find the acceleration of the material point X, which exists at the spatial point $\hat{X}$ at time t. Note that the material velocity field u can be given in terms of the reference or current spatial domain $u = \hat{U}(\hat{X},t) = \hat{u}(\hat{x},t)$. Differentiating u with respect to time gives

$$\frac{du}{dt} = \frac{\partial \hat{U}(\hat{X},t)}{\partial \hat{X}} \cdot \frac{\partial \psi(X,t)}{\partial t} + \frac{\partial \hat{U}(\hat{X},t)}{\partial t}, \tag{2.10}$$

and one can use the fact that

$$\frac{\partial \hat{U}(\hat{X},t)}{\partial \hat{X}} = \frac{\partial \hat{u}(\hat{x},t)}{\partial \hat{x}} \cdot \frac{\partial \hat{\phi}(\hat{X},t)}{\partial \hat{X}}, \tag{2.11}$$

to obtain

$$\frac{du}{dt} = \frac{\partial \hat{u}(\hat{x},t)}{\partial \hat{x}} \cdot \frac{\partial \hat{\phi}(\hat{X},t)}{\partial \hat{X}} \cdot \frac{\partial \psi(X,t)}{\partial t} + \frac{\partial \hat{U}(\hat{X},t)}{\partial t}. \tag{2.12}$$

Next it is shown that the term $\frac{\partial \hat{\phi}(\hat{X},t)}{\partial \hat{X}} \cdot \frac{\partial \psi(X,t)}{\partial t}$ is actually equal to $u - \hat{v}$. In order to see this note that

$$x = \phi(X,t) = \hat{\phi}(\hat{X},t) = \hat{\phi}(\psi(X,t),t) = \hat{x}, \tag{2.13}$$

which can be differentiated with respect to time in order to obtain

$$\frac{\partial \phi(X,t)}{\partial t} = \frac{\partial \hat{\phi}(\hat{X},t)}{\partial t} + \frac{\partial \hat{\phi}(\hat{X},t)}{\partial \hat{X}} \cdot \frac{\partial \psi(X,t)}{\partial t}. \tag{2.14}$$

The left hand side is identified as the current material velocity u of the material point X, whereas the first term on the left hand side can be seen as the current spatial velocity $\hat{v}$ of the spatial point $\hat{X}$. Hence,

$$\frac{\partial \hat{\phi}(\hat{X},t)}{\partial \hat{X}} \cdot \frac{\partial \psi(X,t)}{\partial t} = u - \hat{v}. \tag{2.15}$$

Finally, putting together (2.12) and (2.15) yields

$$\frac{d}{dt}u(x,t) = \dot{u} + \text{grad}_{\hat{x}} u \cdot (u - \hat{v}). \tag{2.16}$$

Here the gradient is specifically identified as being with respect to the current spatial coordinate $\hat{x}$. If the density ρ is added to the above calculations the result is (2.9).

The momentum balance equation in the ALE framework can now be formulated, it reads

$$\int_{\omega_F(t)} (\dot{\rho u}) + \text{grad}\,(\rho u) \cdot (u - \hat{v})\, \mathrm{d}x = \int_{\partial\omega_F(t)} \sigma \cdot n\, \mathrm{d}s + \int_{\omega_F(t)} b\, \mathrm{d}x. \tag{2.17}$$

For a more in depth discussion of the Arbitrary Lagrangian Eulerian framework and balance laws see J. Donea1 and Rodríguez-Ferran [2004].

2.2 Strong Forms of the FSI Equations

In this section the fluid-structure interaction system is presented in three parts, with each part consisting of a chosen partial differential equation. The movement of the fluid in the fluid domain ω_F is modeled by the incompressible Navier-Stokes equations, and the deformation of the structure in ω_S is described by the St.Venant-Kirchhoff equation. Finally, the movement of the fluid domain itself (spatial points), is given by a specially designed 'linear elastic type' equation. The three equations are coupled together at the common boundary of the fluid and solid domains, the FSI interface Γ_{FSI}. The couplings, together with the three equations, and initial and boundary conditions, make up a single system of partial differential equations. The notation for the unknown variables is here fixed for the rest of the thesis. The unknowns are the fluid velocity U_F, fluid pressure P_F, structure displacement D_S, and fluid domain displacement D_F.

2.2.1 Initial conditions

A good starting point for expressing an abstract fluid-structure interaction problem are the initial conditions. These are expressed mathematically as,

$$\begin{aligned} U_{_F}(\cdot\,,0) &= U_{_F}^0, \quad \textit{fluid velocity} \\ P_{_F}(\cdot\,,0) &= P_{_F}^0, \quad \textit{fluid pressure} \\ D_{_S}(\cdot\,,0) &= D_{_S}^0, \quad \textit{structure displacement} \\ D_{_F}(\cdot\,,0) &= D_{_F}^0, \quad \textit{fluid domain displacement} \end{aligned} \tag{2.18}$$

where $U_{_F}^0, P_{_F}^0, D_{_S}^0, D_{_F}^0$ are given functions. In a typical pure fluid dynamics problem modeled by the Navier-Stokes equations the initial fluid pressure is irrelevant, as it is recalculated at each time step independently of the previous pressure. However in the case of fluid-structure interaction the initial fluid pressure is relevant as it effects the displacement of the structure. This will be made more clear in Section 3.3.

2.2.2 Fluid equation in the current domain

The incompressible Navier-Stokes equations are derived from the principles of conservation of momentum and mass, which are given in the current domain in the ALE framework by (2.8) and (2.17). Additional assumptions are made about the behaviour of the fluid that is being modeled. In particular the fluid is assumed to be *Newtonian*. This means that the stress-strain rate is linear. It is also assumed that the fluid experiences no stress when the velocity gradient is equal to zero, implying that the stress-strain curve passes through the origin. Additionally, our fluid is *isotropic*, which means that the stress strain rate is the same in all directions.

The usual Cauchy stress tensor σ is broken up into a volumetric part $\mathcal{P}$, and a deviatoric (shear stress) part $\mathcal{T}$, by the relation

$$\sigma = \mathcal{P} + \mathcal{T}. \tag{2.19}$$

Due to the isotropy of the fluid, the volumetric tensor can be expressed in terms of one variable, the pressure p_F,

$$\mathcal{P} = p_F I. \tag{2.20}$$

Here I represents the identity matrix of the same size as the spatial dimension d, and $p_F = \frac{1}{d}\mathrm{tr}(\sigma)$. $\mathcal{T}$ can now be simply defined by $\mathcal{T} = \mathcal{P} - \sigma_F$.

The assumptions above allow us to represent the shear stress tensor as a function of the strain grad $u_{_F}$, and the kinematic viscosity μ_F

$$\mathcal{T} = 2\mu_F \mathrm{grad}^{\mathbf{s}} u_{_F}. \tag{2.21}$$

Here $\mathrm{grad}^{\mathbf{s}}(\cdot\,) = \frac{1}{2}\left(\mathrm{grad}\,(\cdot\,) + \mathrm{grad}\,(\cdot\,)^\top\right)$ represents the symmetric gradient. Furthermore μ_F is assumed to be constant, but can in general be a function of pressure and temperature for an arbitrary Newtonian fluid.

The final assumption that is made in the fluid model is that of *incompressibility*. This is expressed mathematically as

$$\operatorname{div} u_{_F} = 0. \tag{2.22}$$

As a consequence, the density $\rho_{_F}$ is assumed constant throughout the fluid.

Assuming the above conditions, the Navier-Stokes equations in the current domain are given by

$$\begin{aligned} \mathrm{d}_t(\rho_{_F}, u_{_F}, D_{_F}) - \operatorname{div} \sigma_{_F}(u_{_F}, p_{_F}) = &\quad b_{_F} \quad \text{in } \omega_{_F}(t) \times (0,T], \\ \operatorname{div} u_{_F} = &\quad 0 \quad \text{in } \omega_{_F}(t) \times (0,T], \end{aligned} \tag{2.23}$$

where $\sigma_{_F} = 2\mu_{_F} \operatorname{grad}^{\mathbf{s}} u_{_F} - p_{_F} I$ is the fluid Cauchy stress tensor, and the acceleration term is given by $\mathrm{d}_t(\rho_{_F}, u_{_F}, D_{_F}) = \rho_{_F}(\dot{u}_{_F} + \operatorname{grad} u_{_F} \cdot (u_{_F} - \dot{D}_{_F}))$. The equations are given in so called stress tensor form, as opposed to the Laplacian form obtain by simplifying the stress tensor using the incompressibility constraint. The stress tensor form has the advantage of Neumann boundary terms always representing physical stresses, which is important for the interface conditions. For a discussion of this see Melbø and Kvamsdal [2003].

2.2.3 Fluid equation in the reference domain

In order to formulate a full Newton's method for the FSI problem it is necessary to account for the sensitivity of the fluid variables to the fluid domain displacement $D_{_F}$. This can be accomplished by mapping the Navier-Stokes equation, (2.23), into the reference domain $\Omega_{_F}$, which makes the dependency on $D_{_F}$ clear. The derivation is accomplished by multiple applications of Nanson's formula for boundary integrals:

$$\int_{\partial\omega} x \cdot n \, \mathrm{d}s = \int_{\partial\Omega} Jx \cdot F^{-\top} \cdot N \, \mathrm{d}S, \tag{2.24}$$

and also the usual change of variables rule. In Nanson's formula x is a vector valued function, n the unit outward normal in the current domain, N the unit outward normal in the reference domain, J the Jacobian of the mapping $\phi : \Omega \longrightarrow \omega$, and F the deformation gradient of ϕ.

Firstly the incompressibility term is mapped, using the fact that $u_{_F}(x,t) = U_{_F}(\Phi_{_F}^{-1}x, t)$, the divergence formula, and by switching to integral form and back.

$$\begin{aligned} \int_{\omega_{_F}} \operatorname{div} u_{_F} \, \mathrm{d}x &= \int_{\partial\omega_{_F}} u_{_F} \cdot n \, \mathrm{d}s, \\ &= \int_{\partial\Omega_{_F}} J_{_F} U_{_F} \cdot F_{_F}^{-\top} N \, \mathrm{d}S, \\ &= \int_{\partial\Omega_{_F}} J_{_F} F_{_F}^{-1} U_{_F} \cdot N \, \mathrm{d}S, \\ &= \int_{\Omega_{_F}} \operatorname{Div} (J_{_F} F_{_F}^{-1} U_{_F}) \, \mathrm{d}X = 0. \end{aligned} \tag{2.25}$$

The 'div ' represents the divergence with respect to the current spatial coordinate system, whereas 'Div ' represents the divergence with respect to the reference spatial coordinate system. Furthermore, the acceleration term in the current domain $\mathrm{d}_t(\rho_{_F}, u_{_F}, D_{_F})$ can be transformed into the acceleration term in the reference domain $\mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F})$ by a change of variables

$$\begin{aligned}\int_{\omega_{_F}} \mathrm{d}_t(\rho_{_F}, u_{_F}, D_{_F})\,\mathrm{d}x &= \int_{\omega_{_F}} \rho_{_F}(\dot{u}_{_F} + \operatorname{grad} u_{_F}\cdot(u_{_F} - \dot{D}_{_F}))\,\mathrm{d}x,\\ &= \int_{\Omega_{_F}} J_{_F}\rho_{_F}(\dot{U}_{_F} + \operatorname{Grad} U_{_F}F_{_F}^{-1}\cdot(U_{_F} - \dot{D}_{_F}))\,\mathrm{d}X,\\ &= \int_{\Omega_{_F}} \mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F})\,\mathrm{d}X.\end{aligned} \tag{2.26}$$

Here 'grad ' refers to the gradient with respect to the current spatial coordinate system, and 'Grad ' refers to the gradient with respect to the reference spatial coordinate system. Finally, the current stress tensor $\sigma_{_F}$ can be mapped to the reference stress tensor Σ_F, by the following relation

$$\begin{aligned}\int_{\omega_{_F}} \operatorname{div}\sigma_{_F}\,\mathrm{d}x &= \int_{\partial\omega_{_F}} \sigma_{_F}\cdot n\,\mathrm{d}s,\\ &= \int_{\partial\omega_{_F}} (\mu_{_F}(\operatorname{grad} u_{_F} + \operatorname{grad} u_{_F}^{\top}) - p_{_F}I)\cdot n\,\mathrm{d}s,\\ &= \int_{\partial\Omega_{_F}} J_{_F}(\mu_{_F}(\operatorname{Grad} U_{_F}F_{_F}^{-1} + F_{_F}^{-\top}\operatorname{Grad} U_{_F}^{\top}) - P_{_F}I)F_{_F}^{-\top})\cdot N\,\mathrm{d}s,\\ &= \int_{\partial\Omega_{_F}} J_{_F}\Sigma_{_F}F_{_F}^{-\top}\cdot N\,\mathrm{d}s,\\ &= \int_{\Omega_{_F}} J_{_F}\operatorname{div}(\Sigma_{_F}F_{_F}^{-\top})\,\mathrm{d}X\end{aligned} \tag{2.27}$$

Here the two stress tensors are related by the so called *Piola transform* $\sigma_{_F} = J_{_F}\Sigma_{_F}\cdot F_{_F}^{-\top}$.

Putting together (2.25), (2.26) and (2.27), the incompressible Navier-Stokes equations can be formulated in the reference domain.

$$\begin{aligned}\mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F}) - \operatorname{Div}(J_{_F}\,\Sigma_{_F}(U_{_F}, P_{_F}, D_{_F})\cdot F_{_F}^{-\top}) &= B_{_F} \qquad \text{in } \Omega_{_F}\times(0,T],\\ \operatorname{Div}(J_{_F}\,F_{_F}^{-1}\cdot U_{_F}) &= 0 \qquad \text{in } \Omega_{_F}\times(0,T].\end{aligned} \tag{2.28}$$

Here the current and reference body forces are related by

$$B_{_F} = J_{_F}b_{_F}. \tag{2.29}$$

The operators $\mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F})$ and $\Sigma_{_F}(U_{_F}, P_{_F}, D_{_F})$, which were defined im-

plicitly in (2.26) and (2.27), are here defined as

$$\begin{aligned} \mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F}) &= J_{_F}\rho_{_F}(\dot{U}_{_F} + \mathrm{Grad}\, U_{_F}F_{_F}^{-1} \cdot (U_{_F} - \dot{D}_{_F})), \\ \Sigma_{_F}(U_{_F}, P_{_F}, D_{_F}) &= \mu_{_F}(\mathrm{Grad}\, U_{_F}F_{_F}^{-1} + F_{_F}^{-\top}\mathrm{Grad}\, U_{_F}^{\top}) - P_{_F}I. \end{aligned} \tag{2.30}$$

2.2.4 Structure equation

For the structure motion, we consider the St.Venant-Kirchhoff hyperelastic model. This model represents the natural extension of linear elastic theory into the nonlinear regime. The model is fairly simple, so that relatively few parameters are added to the FSI system, but still presents an interesting computational challenge in the form of the nonlinearity.

Similar to the Navier-Stokes equations, the hyperelastic equations are derived from mass and momentum continuity. For the structure these were formulated previously in Section 2.1.2 in the Lagrangian framework and over the reference domain. The stress-strain relationship of a St.Venant-Kirchoff hyperelastic material is given by the second Piola-Kirchoff stress Σ_S,

$$\Sigma_S = F_{_S} \cdot (2\mu_{_S}\, E_{_S} + \lambda_{_S}\, \mathrm{tr}\,(E_{_S})I). \tag{2.31}$$

Where $E_{_S}$ is the Green–Lagrange strain tensor

$$E_{_S} = \tfrac{1}{2}(F_{_S}^{\top} \cdot F_{_S} - I), \tag{2.32}$$

and $\mu_{_S}, \lambda_{_S}$ are the Lamé constants, which further model how the material deforms under stress. In the definition of $E_{_S}$, $F_{_S}$ is the deformation gradient, given previously in section 2.1.1.

The hyperelastic equation reads, find the displacement $D_{_S}$, such that

$$\mathrm{D}_t^2(\rho_{_S} D_{_S}) - \mathrm{Div}\, \Sigma_{_S}(D_{_S}) = B_{_S} \quad \text{in } \Omega_{_S} \times (0, T]. \tag{2.33}$$

Here, $B_{_S}$ is a given reference body force. The acceleration term is given by $\mathrm{D}_t^2(\rho_{_S} D_{_S}) = \rho_{_S}\, \ddot{D}_{_S}$, where $\rho_{_S}$ is the constant reference structure density. By feature of the Lagrangian framework the mass continuity equation can be ignored, as all of the matter present in the structure is tracked throughout the entire simulation.

2.2.5 Fluid domain equation

The deformation of the fluid domain is given on the fluid-structure interface $\Gamma_{_{FSI}}$ by the structure displacement $D_{_S}$. Due to the finite element formulation of the FSI problem it is necessary to extend the mapping $D_{_S}|_{\Gamma_{_{FSI}}}$ into the fluid domain in such a way that the image of the mesh in the current domain is of a sufficient quality for the simulation of the fluid equations. Especially one wants to avoid the creation of thin triangles, or any overlap in the mesh, as these can result in singular matrices during assembly.

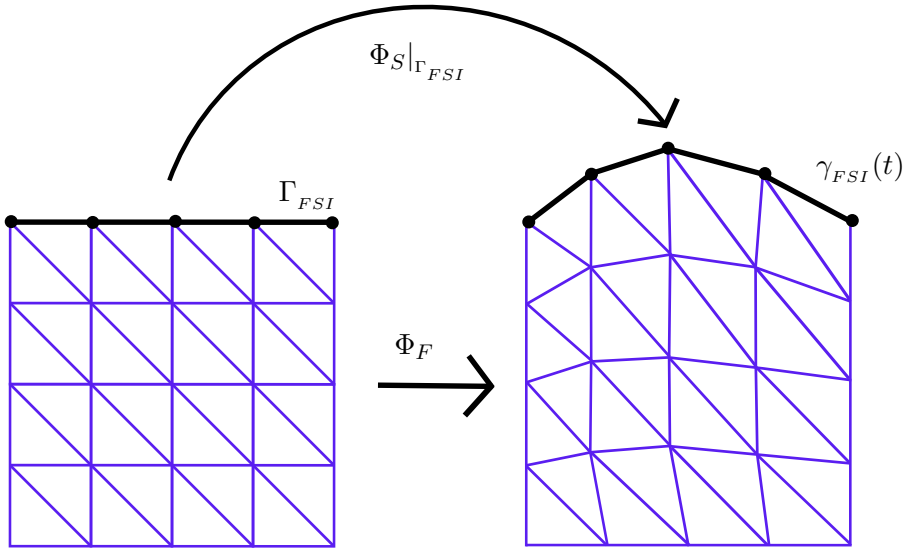

Figure 2.5: Illustration of the fluid domain equation. The unknown value of this equation is $D_{_F}$, which is related to the fluid domain mapping by $\Phi_F(X) = X + D_{_F}(X)$. The displacement along the FSI interface (shown in black) is given by the structure equation. The displacement is smoothly extended into the fluid domain in order to preserve the quality of the image of the mesh under Φ_F, shown on the right.

In Selim [2011] an additional differential equation for the movement of the fluid mesh was introduced, called the mesh equation. In this work the same equation is used, but with the purpose of calculating the value of $D_{_F}$ inside the fluid domain, rather than for the movement of the mesh itself.

The fluid domain equation can be thought of as a parabolic linear elasticity equation. It is given by

$$\dot{D}_{_F} - \text{Div}\, \Sigma_{_{FD}}(D_{_F}) = 0 \quad \text{in } \Omega_{_F} \times (0, T], \tag{2.34}$$

where the fluid domain stress tensor can be written as

$$\Sigma_{_{FD}}(D_{_F}) = 2\mu_{_{FD}}\, \text{Grad}^{\text{s}} D_{_F} + \lambda_{_{FD}} \text{tr}(\text{Grad}\, D_{_F}) I, \tag{2.35}$$

for positive parameters $\lambda_{_{FD}}$ and $\mu_{_{FD}}$. The fluid domain equation (2.34), being linear, is easy to solve and does not unduly complicate the FSI system.

An alternative fluid domain equation can be found in Dettmer and Perić [2006] which minimizes the sum of the ratios of the inner and outer circles of simplicial elements.

2.2.6 Equation couplings

The three subproblems of the FSI problem, fluid, structure, and fluid domain, are coupled on the FSI interface $\Gamma_{_{FSI}}$. In particular, the following conditions should hold on $\Gamma_{_{FSI}}$:

$$\begin{array}{rcll} U_{_F} & = & \dot{D}_{_S} & \textit{kinematic continuity}, \\ J_{_F} \Sigma_{_F} \cdot F_{_F}^{-\top} \cdot N_F & = & -\Sigma_{_S} \cdot N_S & \textit{stress continuity}, \\ D_{_F} & = & D_{_S} & \textit{domain continuity}. \end{array} \tag{2.36}$$

The kinematic continuity condition ensures that the material of the fluid and structure are stuck together, the stress continuity condition ensures that the FSI model obeys Newton's third law, and the domain continuity condition ensures that the fluid and structure domains remain connected. The relationship $N_S = -N_F$ can be used to reformulate the stress continuity condition in terms of a single unit outer normal.

2.2.7 Boundary Conditions

The boundary conditions for the subproblems are given on $\Gamma_{_{FSI}}$ by the equation couplings (2.36). However, boundary conditions for the exterior domain boundaries are also needed in order to obtain a well posed problem. These are listed in Table 2.1.

Dirichlet type boundary conditions (including fluid velocity no slip) are used in this thesis to set a velocity or displacement along the boundary, whereas the Neumann boundary conditions are used to prescribe a normal

Table 2.1: Boundary conditions that are considered in this thesis. G_F, G_S denote Neumann boundary value functions, and $\overline{P_F}, \overline{D_S}$ denote Dirichlet boundary value functions

Name	**Symbol**	**Condition**
Fluid velocity no slip	$\Gamma_{F,0}$	$U_F = 0,$
Fluid velocity Neumann	$\Gamma_{F,N}$	$\Sigma_F \cdot N_F = G_F,$
Fluid do nothing	Γ_{DN}	$\operatorname{Grad} U_F = 0,$
Fluid pressure Dirichlet	$\Gamma_{P,D}$	$P_F = \overline{P}_F,$
Structure displacement Dirichlet	$\Gamma_{S,D}$	$D_S = \overline{D}_S,$
Structure Neumann	$\Gamma_{S,N}$	$\Sigma_S \cdot N_S = G_S,$
Fluid domain Dirichlet	$\Gamma_{FD,0}$	$D_F = 0.$

force. In the upcoming weak form of the FSI problem given in section 2.3, variational forms are given corresponding to the Neumann conditions above.

The fluid do nothing condition is used where a fully developed fluid flow is assumed, i.e. the velocity profile does not change outside of the computational domain. A typical place to use such a boundary condition is at the outflow of a section of pipe, as it guarantees that the fluid leaves the pipe section as if the pipe continued, instead of spraying out as if the pipe suddenly ended.

Pressure Dirichlet boundaries are optional, and can be used to make the pressure unique in the case that all of the fluid velocity boundaries are of Dirichlet type. The Navier-Stokes equations themselves cannot provide a unique pressure, as only the gradient of the pressure appears in the strong form, and not the pressure itself. Pressure uniqueness can also be obtained in the pure velocity Dirichlet case by enforcing $0 = \int_{\Omega_F} p \, \mathrm{d}x$. For more information regarding the proper use of boundary conditions in fluid dynamics see Rannacher [1999].

The various sub boundaries of Table 2.1 should cover up their respective domain boundaries without overlapping. In particular,

$$\partial\Omega_F = \Gamma_{FSI} \cup \Gamma_{F,0} \cup \Gamma_{F,N} \cup \Gamma_{out}, \tag{2.37}$$

is desired for the fluid velocity boundaries. Also, the various structure boundaries should cover the structure domain boundary

$$\partial\Omega_S = \Gamma_{FSI} \cup \Gamma_{S,D} \cup \Gamma_{S,N}. \tag{2.38}$$

In this thesis the fluid domain boundary is fixed outside of the interface

Γ_{FSI}, meaning that

$$\partial\Omega_F = \Gamma_{FSI} \cup \Gamma_{FD,0}. \tag{2.39}$$

2.2.8 Strong form of the FSI System

The three partial differential equations (2.28), (2.33), (2.34), together with the couplings over the fluid-structure interface (2.36), a set of boundary conditions from Table 2.1, and initial conditions constitute a fully specified fluid-structure interaction problem that can be solved using the code developed as part of this master thesis. The question as to the existence and uniqueness of the solutions to the FSI system presented here in strong form is open. Any theorem regarding this issue would probably have to address the existence and uniqueness of solutions to the Navier-Stokes equations. This is a difficult problem, and is also one of the seven Millenium challenges put forth by the Clay Mathematics institute in the year 2000.

2.3 Weak Form of the FSI Equations

A weak form of the FSI problem can be formulated by the standard technique of multiplying the strong forms by test functions, integrating in time and space, and using integration by parts in order to balance derivatives. Non-interface Dirichlet boundary conditions are enforced strongly, which means that they enter into the description of the trial and test function spaces. Trial functions agree with these conditions, whereas test functions vanish over the corresponding Dirichlet boundaries. Non-interface Neumann boundary terms, resulting from integration by parts, are listed in the sections where they arise.

Three new unknowns are introduced in the weak formulation, (U_S, L_U, L_D). U_S represents the structure velocity, and (L_U, L_D) are Lagrange multipliers corresponding to the velocity and displacement equation couplings. In total seven trial functions $(U_F, P_F, L_U, D_S, U_S, D_F, L_D)$ and seven test functions $(v_F, q_F, m_U, c_S, v_S, c_F, m_D)$ are used in formulating the weak FSI problem. The notation $\langle \cdot\,,\, \cdot \rangle_F$ is used to denote the L^2 inner product over the fluid domain, and $\langle \cdot\,,\, \cdot \rangle_S$ the L^2 inner product over the structure domain. Boundary inner products are denoted by a subscript containing the appropriate boundary.

2.3.1 Weak form of the fluid equations

In formulating the weak form of the fluid equations we consider the two trial functions $(U_{_F}, P_{_F})$ with corresponding trial spaces

$$\begin{aligned} U_{_F} \in V_{_F} &= \left\{ \begin{array}{l} v \in L^2(0,T;[H^1(\Omega_{_F})]^d : v(\cdot\,,0) = U^0_{_F}, v|_{\Gamma_{F,0}} = 0 \\ \dot{v} \in L^2(0,T;[H^{-1}(\Omega_{_F})]^d \end{array} \right\}, \\ P_{_F} \in Q_{_F} &= \left\{ \begin{array}{l} v \in L^2(0,T;L^2(\Omega_{_F})) : v(\cdot\,,0) = P^0_{_F}, v|_{\Gamma_{P,D}} = \overline{P}_F \\ \dot{v} \in L^2(0,T;[H^{-1}(\Omega_{_F})] \end{array} \right\}. \end{aligned} \tag{2.40}$$

where d is the spatial dimension.

If we examine the definition of $V_{_F}$ we see that a condition has been placed on the time derivative, namely $\dot{v} \in L^2(0,T;[H^{-1}(\Omega_{_F})]^d$. Without this condition functions in $V_{_F}$ would only be L^2 in time, meaning that our pointwise in time initial condition $v(\cdot\,,0) = U^0_{_F}$ would not make sense. With the condition however, one can see from theorem 3 on page 303 of Evans [2010], that unique continuous in time version of the functions in $V_{_F}$ exist, which makes the initial condition meaningful. Similar conditions are present in other spaces which include an initial condition.

The test spaces corresponding to the test functions $(v_{_F}, q_{_F})$ are denoted $(\hat{V}_{_F}, \hat{Q}_{_F})$, and are defined analogously to the trial spaces, except that the initial and boundary conditions, including $\Gamma_{_{FSI}}$, are homogenized, i.e. the value over the boundary is set to 0

$$\begin{aligned} v_{_F} \in \hat{V}_{_F} &= \left\{ \begin{array}{l} \in L^2(0,T;[H^1(\Omega_{_F})]^d : v(\cdot\,,0) = 0, v|_{\Gamma_{F,0}} = 0, v|_{\Gamma_{FSI}} = 0 \\ \dot{v} \in L^2(0,T;[H^{-1}(\Omega_{_F})]^d \end{array} \right\}, \\ q_{_F} \in \hat{Q}_{_F} &= \left\{ \begin{array}{l} v \in L^2(0,T;L^2(\Omega_{_F})) : v(\cdot\,,0) = 0, v|_{\Gamma_{P,D}} = 0 \\ \dot{v} \in L^2(0,T;[H^{-1}(\Omega_{_F})] \end{array} \right\}. \end{aligned} \tag{2.41}$$

The residual corresponding to the weak form of the fluid equations (2.28) $R_F : (\hat{V}_{_F} \times \hat{Q}_{_F} \times V_{_F} \times Q_{_F} \times C_{_F}) \longrightarrow \mathbb{R}$ is given by

$$\begin{aligned} R_F(v_{_F}, q_{_F}, P_{_F}; U_{_F}, D_{_F}) = &\int_0^T \langle v_{_F},\ \mathrm{D}_t(\rho_{_F}, U_{_F}, D_{_F})\rangle_{_F}\ \mathrm{d}t, \\ &+ \int_0^T \left\langle \mathrm{Grad}\, v_{_F},\ J_{_F}\ \Sigma_{_F}(U_{_F}, P_{_F}, D_{_F}) F_{_F}^{-\top} \right\rangle_{_F}\ \mathrm{d}t, \\ &+ \int_0^T \left\langle q_{_F},\ \mathrm{Div}\,(J_{_F} F_{_F}^{-1} U_{_F})\right\rangle_{_F},\ \mathrm{d}t \\ &- \int_0^T \langle v_{_F},\ B_{_F}\rangle_{_F}\ \mathrm{d}t - \int_0^T \langle v_{_F},\ G_F\rangle_{\Gamma_{F,N}}. \end{aligned} \tag{2.42}$$

If the fluid do nothing boundary condition is used, it can be implemented weakly by setting $\mathrm{Grad} U_{_F} = 0$ in the boundary term along the corresponding

boundary. The condition is then imposed by adding

$$\int_0^T \left\langle v_{_F},\ \mu_{_F}(F_{_F}^{-\top}\operatorname{Grad} U_{_F}^{\top}) - P_{_F} I \right\rangle_{\Gamma_{DN}} \,\mathrm{d}t, \tag{2.43}$$

to R_F.

2.3.2 Weak form of the structure equations

To simplify the formulation of a time-stepping scheme, a variable $U_{_S} = \dot{D}_{_S}$ for the structure velocity is introduced. This allows the structure equation (2.33) to be rewritten as a system that is first order in time:

$$\begin{aligned} \rho_{_S}\dot{U}_{_S} - \operatorname{Div}\Sigma_{_S}(D_{_S}) &= B_{_S} \qquad \text{in } \Omega_{_S}\times(0,T], \\ \dot{D}_{_S} &= U_{_S} \qquad \text{in } \Omega_{_S}\times(0,T]. \end{aligned} \tag{2.44}$$

From this system the structure weak form is derived. The trial spaces corresponding to the variables $(D_{_S}, U_{_S})$ are

$$\begin{aligned} D_{_S} \in C_{_S} &= \left\{ \begin{array}{l} v \in L^2(0,T[H^1(\Omega_{_S})]^d) : v(\cdot\,,0) = D_{_S}^0, v|_{\Gamma_{S,D}} = \overline{D}_S \\ \dot{v} \in L^2(0,T[H^{-1}(\Omega_{_S})]^d \end{array} \right\}, \\ U_{_S} \in V_{_S} &= \{v \in L^2(0,T;[L^2(\Omega_{_S})]^d)\}. \end{aligned}$$

The test spaces corresponding to the test functions $(c_{_S}, v_{_S})$ are $(\hat{C}_{_S}, \hat{V}_{_S})$, and are the initial and boundary condition homogeneous versions of the trial spaces

$$\begin{aligned} c_{_S} \in \hat{C}_{_S} &= \left\{ \begin{array}{l} v \in L^2(0,T[H^1(\Omega_{_S})]^d) : v(\cdot\,,0) = 0, v|_{\Gamma_{S,D}} = 0 \\ \dot{v} \in L^2(0,T[H^{-1}(\Omega_{_S})]^d \end{array} \right\}, \\ v_{_S} \in \hat{V}_{_S} &= \{v \in L^2(0,T;[L^2(\Omega_{_S})]^d\}. \end{aligned}$$

The residual corresponding to the weak form of the structure equations $R_S : (\hat{C}_{_S}\times\hat{V}_{_S}\times C_{_S}\times V_{_S}) \longrightarrow \mathbb{R}$ is given by

$$\begin{aligned} R_S(c_{_S}, v_{_S}, U_{_S}; D_{_S}) = &\int_0^T \left\langle c_{_S},\ \rho_{_S}\dot{U}_{_S} \right\rangle_{_S} \,\mathrm{d}t + \int_0^T \langle \operatorname{Grad} c_{_S},\ \Sigma_{_S}(D_{_S})\rangle_{_S} \,\mathrm{d}t, \\ &+ \int_0^T \left\langle v_{_S},\ \dot{D}_{_S} - U_{_S} \right\rangle_{_S} \,\mathrm{d}t - \int_0^T \langle c_{_S},\ B_{_S}\rangle_{_S} \,\mathrm{d}t, \\ &- \int_0^T \langle c_{_S},\ G_S\rangle_{\Gamma_{S,N}} \,. \end{aligned} \tag{2.45}$$

2.3.3 Weak form of the fluid domain equations

In contrast to the weak forms of the fluid and structure, the fluid domain weak form only contains a single trial function $D_{_F}$, with corresponding trial function space $C_{_F}$

$$D_{_F} \in C_{_F} = \left\{ \begin{array}{l} v \in L^2(0,T[H^1(\Omega_{_F})]^d) : v(\cdot\,,0)|_{\Gamma_{_{FSI}}} = U_{_S}^0|_{\Gamma_{_{FSI}}}, v|_{\Gamma_{_{FD,0}}} = 0 \\ \dot{v} \in L^2(0,T[H^{-1}(\Omega_{_F})]^d) \end{array} \right\}. \tag{2.46}$$

The single test function $c_{_F}$ is a member of the test function space $\hat{C}_{_F}$

$$c_{_F} \in \hat{C}_{_F} = \{v \in L^2(0,T[H^1(\Omega_{_F})]^d) : v|_{\partial\Omega_{_F}} = 0\}. \tag{2.47}$$

The residual corresponding to the weak form of the fluid domain equations $R_{FD} : (\hat{C}_{_F} \times C_{_F}) \longrightarrow \mathbb{R}$ is given by

$$R_{FD}(c_{_F}, D_{_F}) = \int_0^T \left\langle c_{_F},\ \dot{D}_{_F} \right\rangle_{_F} \mathrm{d}t + \int_0^T \langle \mathrm{Grad}^{\mathbf{s}} c_{_F},\ \Sigma_{_{FD}}(D_{_F}) \rangle_{_F}\ \mathrm{d}t. \tag{2.48}$$

Due to the symmetry of the fluid domain stress tensor $\Sigma_{_{FD}}$, the test function gradient $\mathrm{Grad}\, c_{_F}$ can also be made symmetric in the second term of R_{FD}. This is due to the identity, $A : B = \frac{1}{2}(B + B^\top) : A$, which holds for second order tensors when A is symmetric.

2.3.4 Weak form of the equation couplings

One of the key challenges in solving FSI problems is the enforcement of the interface conditions. This is accomplished here by implementing the required conditions 2.36 in a weak sense, through the addition of an interface residual $R_{\Gamma_{_{FSI}}}$ to the weak forms.

The second interface condition, corresponding to the stress continuity, is the most straightforward to implement. If the structure stress tensor term in the strong form (2.33) is integrated by parts, it yields

$$\langle c_{_S},\ -\mathrm{Div}\ \Sigma_{_S} \rangle_{_S} = \langle \mathrm{Grad}\, c_{_S},\ \Sigma_{_S} \rangle_{_S} - \langle c_{_S},\ \Sigma_{_S} \cdot N_S \rangle_{\Gamma_{_{FSI}}} - \langle c_{_S},\ \Sigma_{_S} \cdot N_S \rangle_{\Gamma_{_{S,N}}}. \tag{2.49}$$

The last term can be recognized as being equal to the Neumann value G_S, which was included in the presentation of the structure residual (2.45). Furthermore, the middle term can be equated to the fluid stress on $\Gamma_{_{FSI}}$ using stress continuity. This means that the inclusion of the term $\int_0^T \left\langle c_{_S},\ J_{_F} \Sigma_{_F} \cdot F_{_F}^{-\top} \cdot N_F \right\rangle_{\Gamma_{_{FSI}}} \mathrm{d}t$, in the residual $R_{\Gamma_{_{FSI}}}$, is enough to enforce the required stress continuity.

The two "Dirichlet type" couplings, that of mesh and kinematic continuity, are a little more tricky to implement. They do not show up directly in a weak form, and they cannot be implemented strongly as Dirichlet boundary

conditions since their values are not known before hand. Instead the technique of Lagrange multipliers is used to implement the conditions in a weak sense. Implementing Lagrange multipliers involves the addition of two new trial functions

$$L_{_D} \in M_{_D} = \{m \in L^2(0,T;[L^2(\Gamma_{_{FSI}})]^d)\}, \tag{2.50}$$
$$L_{_U} \in M_{_U} = \{m \in L^2(0,T;[L^2(\Gamma_{_{FSI}})]^d)\}, \tag{2.51}$$

with corresponding test functions

$$m_{_D} \in \hat{M}_{_D} = M_{_D}, \tag{2.52}$$
$$m_{_U} \in \hat{M}_{_U} = M_{_U}. \tag{2.53}$$

The enforcement of the desired interface conditions, $U_{_F} - U_{_S} = 0 = D_{_F} - D_{_S}$, is achieved through the addition of the terms

$$\int_0^T \langle m_{_U},\ U_{_F} - U_{_S} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t + \int_0^T \langle m_{_D},\ D_{_F} - D_{_S} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t, \tag{2.54}$$
$$\tag{2.55}$$

to the interface residual. Finally, two more terms are added which involve the Lagrange multiplier trial functions

$$\int_0^T \langle v_{_F},\ L_{_U} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t + \int_0^T \langle c_{_F},\ L_{_D} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t. \tag{2.56}$$

These last two terms balance out the number of constraints and unknowns in the system, and specify that the kinematic continuity condition is to be applied to the fluid velocity, and the domain continuity condition to the fluid domain displacement. For more details regarding the use of Lagrange multipliers see I.Babuška [1973].

The interface residual $R_{\Gamma_{_{FSI}}} : (\hat{V}_{_F} \times \hat{C}_{_S} \times \hat{C}_{_F} \times \hat{M}_{_U} \times \hat{M}_{_D} \times V_{_F} \times Q_{_F} \times C_{_S} \times V_{_S} \times C_{_F} \times M_{_U} \times M_{_D}) \longrightarrow \mathbb{R}$ can now be given. The weak forms corresponding to $R_{\Gamma_{_{FSI}}}(v_{_F}, c_{_S}, c_{_F}, m_{_U}, m_{_D}, U_{_F}, P_{_F}, D_{_S}, U_{_S}, D_{_F}, L_{_U}, L_{_D})$ are,

$$\begin{aligned} R_{\Gamma_{_{FSI}}} = &\int_0^T \langle v_{_F},\ L_{_U} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t, + \int_0^T \langle m_{_U},\ U_{_F} - U_{_S} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t, \\ &+ \int_0^T \langle c_{_F},\ L_{_D} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t, + \int_0^T \langle m_{_D},\ D_{_F} - D_{_S} \rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t, \\ &+ \int_0^T \left\langle c_{_S},\ J_{_F}\ \Sigma_{_F} \cdot F_{_F}^{-\top} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}}\ \mathrm{d}t. \end{aligned} \tag{2.57}$$

2.3.5 Weak Form of the FSI System

Using the residuals from the previous sections, the weak form of the entire FSI system can be formulated. It reads, find

$$(U_{_F}, P_{_F}, L_{_U}, D_{_S}, U_{_S}, D_{_F}, L_{_D}) \in (V_{_F} \times Q_{_F} \times M_{_U} \times C_{_S} \times V_{_S} \times C_{_F} \times M_{_D}) \tag{2.58}$$

such that the equality

$$R = R_F + R_S + R_{FD} + R_{\Gamma_{FSI}} = 0, \tag{2.59}$$

holds for all test functions

$$(v_F, q_F, m_U, c_S, v_S, c_F, m_D) \in (\hat{V}_F \times \hat{Q}_F \times \hat{M}_U \times \hat{C}_S \times \hat{V}_S \times \hat{C}_F \times \hat{M}_D). \tag{2.60}$$

It is worth noting that the nonlinearity in the system lies chiefly in the effect of the fluid domain displacement D_F on the fluid residual R_F. The two other nonlinear terms are the convective acceleration in the fluid $\operatorname{Grad} U_F F_F^{-1} \cdot (U_F - \dot{D}_F))$, which is nonlinear in U_F, and the structure stress tensor Σ_S, which is nonlinear in D_S.

Chapter 3

FINITE ELEMENT FORMULATION OF THE FSI EQUATIONS

In this chapter we present a discretized formulation of the FSI partial differential equation system that was introduced in Chapter 2. We begin with a general introduction to the finite element method, time discretization, and Newton's method, and then apply these techniques to our FSI system.

3.1 The Finite Element Method

The finite element method (FEM) is a framework for the numerical solution of partial differential equations. Since the vast majority of these equations are either very difficult or impossible to solve analytically, numerical solutions provide a necessary alternative. As a solution method, FEM has become very widely accepted by scientists and engineers due to it's wide range of application.

The foundations of the finite element method were developed by Galerkin (1915), who formulated a general method for solving differential equations. This formulation was closely related to the variational principles of Leibniz, Euler, Lagrange, Hamilton, Rayleigh and Ritz (Ritz, 1909). The modern formulation of the finite element method was developed by Alexander Hrennikoff (1941), and Richard Courant (1942), and was applied to structural mechanics problems in the 1950's. Since then FEM grown into many other areas e.g. magnetics and fluid dynamics.

The formulation of a finite element method is carried out in general using four stages, the *strong problem*, *weak problem*, *finite element formulation* and finally *algorithm*. These stages are described below for an abstract model problem.

3.1.1 The strong problem

A strong problem is a partial differential equation that is formulated in local form, that is for every point in the solution domain, and is often the starting point of a finite element method. Fundamental physical principles and modeling assumptions are often the inspiration for a strong problem.

We here consider an abstract PDE problem that is of the same class as that of the FSI system, namely nonlinear and time dependent. The abstract problem reads, find the unknown vector valued function $u : \Omega \times [0, T] \longrightarrow \mathbb{R}^n$ such that

$$\dot{u} + Au = f \text{ in } \Omega \times [0, T]. \tag{3.1}$$

Here $\Omega \subset \mathbb{R}^n$ is an open bounded domain, $\dot{u} = \frac{\partial u}{\partial t}$ the partial derivative of u with respect to time, and A a spatial differential operator. The left hand side of the equation, f, is a given function, often representing external forces. The unknown variable u represents the quantity of interest in the system that is modeled. It could for example be a velocity, a set of chemical concentrations, or in general a vector representing the entire state of a physical system. In addition to a defining equation a strong problem typically requires a set of boundary and initial conditions.

3.1.2 The weak problem

A weak problem is formulated from the strong problem by multiplying the local differential equation with an arbitrary test function v and integrating the new equation in time and space. For the abstract problem (3.1), the result is

$$\int_0^T \langle \dot{u},\ v \rangle + \langle Au,\ v \rangle \ \mathrm{d}t = \int_0^T \langle f,\ v \rangle \ \mathrm{d}t. \tag{3.2}$$

Here $\langle \cdot\ ,\ \cdot \rangle$ represents the L^2 inner product over the domain Ω. If possible, integration by parts is used in order to move derivatives from u to v, which results in an equation that demands less smoothness of u. Any solution that satisfies the strong form will satisfy the weak form due to the generality of the operations used to derive the weak form. The opposite however, is not always true, as a weak solution may not have enough smoothness to be tested by the strong form.

The abstract weak form may be reformulated as an equation involving a single differential operator $a : \hat{V} \times V \longrightarrow \mathbb{R}$,

$$a(v; u) = L(v). \tag{3.3}$$

Here we seek the value of $u \in V$ such that the equation holds for all $v \in \hat{V}$. The function space V is called the trial space, and the function space $\hat{V}$ is called the test space. The notation $a(\ \cdot\ ;\ \cdot\ )$ indicates an operator that is linear in the arguments preceding the semicolon, and nonlinear in the

arguments after the semicolon. The right hand side of the equation is the linear functional $L(v) = \int_0^T \langle f,\ v\rangle\ \mathrm{d}t$. For certain problems the existence and uniqueness of weak solutions can be determined. For an example of this please see Evans [2010, chapter 6] with regards to elliptical problems.

When formulating a Newton's method it is often convenient to consider the so called residual form $R : (\hat{V} \times V) \longrightarrow \mathbb{R}$,

$$R(v; u) = a(v; u) - L(v) = 0. \tag{3.4}$$

3.1.3 The finite element formulation

Time Discretization

The abstract weak form (3.3) is first order in time, and can be time discretized as if it were a system of ordinary differential equations. One such time discretization scheme is the cG(1), or Crank-Nicolson scheme. In formulating the cG(1) scheme the time interval $[0, T]$ is divided into sub intervals $I_n = (t_{n-1}, t_n]$ such that $\bigcup I_n = [0, T]$. Furthermore the trial function space V is restricted to V^k, the subspace of trial functions that are piecewise discontinuous and polynomial in time . That is

$$V^k = \{v \in V : v|_{I_n} \in \mathcal{P}^q(I_n)\}, \tag{3.5}$$

where $\mathcal{P}^q$ denotes a space of qth order polynomials. The cG(1) time stepping scheme corresponds to the choice $q = 1$. By considering test functions that are constant in time over a single time interval and zero everywhere else, i.e.

$$\hat{V}^k = \{v \in \hat{V} : v|_{I_n} \in \mathcal{P}^0(I_N)\}, \tag{3.6}$$

we obtain the set of systems

$$\int_{I_n} \langle v,\ \dot{u_k}\rangle + a^t(v; u_k)\ \mathrm{d}t = \int_{I_n} \langle v,\ f\rangle\ \mathrm{d}t, \tag{3.7}$$

where $u_k \in V^k$ and $a^t(v; u) = \langle v,\ Au\rangle$. Since u_k is first order in time we can evaluate the time derivative term exactly

$$\int_{I_n} \langle v,\ \dot{u_k}\rangle\ \mathrm{d}t = \left\langle v,\ \frac{u^n - u^{n-1}}{k_n}\right\rangle. \tag{3.8}$$

Here the superscript n denotes the value of the function at the n^{th} time level, $u^n = u_k(t = t_n)$, and k_n the value of the n^{th} time step, $k_n = t_n - t_{n-1}$. If the operator a^t is linear, than it is possible to evaluate the term $\int_{I_n} a^t(v, u_k)\ \mathrm{d}t$ exactly using the assumption that u_k is linear and the midpoint rule

$$\int_{I_n} a^t(v, u_k)\ \mathrm{d}t = a^t\left(v, \frac{u^n + u^{n-1}}{2}\right). \tag{3.9}$$

If a^t is not linear than the midpoint rule doesn't hold, but an approximation can still be made

$$\int_{I_n} a^t(v; u_k)\ \mathrm{d}t \approx a^t\left(v; \frac{u^n + u^{n-1}}{2}\right), \tag{3.10}$$

yielding a sequence of spatial partial differential equations that may be solved recursively for u^n. The nth equation in the sequence is given by

$$\left\langle v,\ \frac{u^n - u^{n-1}}{k_n}\right\rangle + a^t\left(v; \frac{u^n + u^{n-1}}{2}\right) = b^n, \tag{3.11}$$

where $b^n = \left\langle v,\ \frac{f^n + f^{n-1}}{2}\right\rangle$.

Spatial Discretization

The first step in creating a finite element formulation is the creation of a mesh, which is a division of the solution domain Ω into a set of non-overlapping simplicial cells, $\mathcal{K}_i$ such that $\bigcup \mathcal{K}_i = \Omega$. In one dimension the cells are intervals, in two triangles, and in three tetrahedra. For dimensions higher than one, other shapes, such as hexagons or cubes, are possible. An example mesh is shown in Figure 3.1.

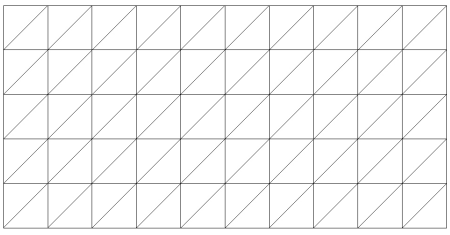

Figure 3.1: Illustration of a simple 2D mesh defined over a rectangle.

Over an arbitrary cell in the mesh, $\mathcal{K}_i$, it is possible to define a local discrete function space, V_i^h, along with a set of basis functions for that space. The global discrete function spaces $(V^h, \hat{V}^h)$ are formed by piecing together the local function spaces, that is

$$\begin{aligned} V^h &= \{u \in V : u|_{\mathcal{K}_i} \in V_i^h \quad \forall i\}, \\ \hat{V}^h &= \{u \in \hat{V} : u|_{\mathcal{K}_i} \in \hat{V}_i^h \quad \forall i\}. \end{aligned} \tag{3.12}$$

A simple and popular choice of discrete space is the cG(1) space, which is built from Lagrange elements of degree 1. The local function spaces in the cG(1) formulation are simply first order polynomial spaces. In the case of a

2D mesh the global cG(1) basis functions are tent shaped functions having the value one over a single mesh vertex, and value zero over neighbouring vertices (see Figure 3.2).

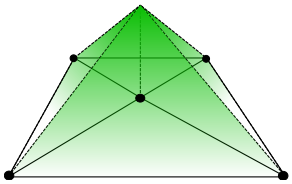

Figure 3.2: Illustration of a global basis function over a 2D mesh for a linear Lagrange element.

The main advantage of the finite element formulation is that it allows the unknown discrete function to be written as a linear combination of basis functions

$$u^h = \sum_{i=1}^{N} \tilde{U}_i(t)\phi_i(x), \tag{3.13}$$

where $\tilde{U}_i(t)$ are the time dependent expansion coefficients of the linear combination, and N is the dimension of the global function space. The discrete test function v^h also has such a representation. If we replace (u, v) with (u^h, v^h) in equation (3.4) we obtain

$$\begin{aligned}
0 &= R(v^h; u^h), \\
&= R\left(\sum_{j=1}^{N} \phi_i(x); \sum_{i=1}^{N} \tilde{U}_i(t)\phi_i(x)\right), \\
&= \int_0^T \sum_{i,j=1}^{N} \frac{\partial \tilde{U}_i(t)}{\partial t} \langle \phi_j,\ \phi_i \rangle + \left\langle \phi_j,\ A\left(\sum_{i=1}^{N} \tilde{U}_i(t)\phi_i\right)\right\rangle - \langle f,\ \phi_j \rangle \ \mathrm{d}t, \\
&= r(\tilde{U}(t)).
\end{aligned} \tag{3.14}$$

Here the coefficients of v^h have been divided out of the equation using the linearity of R in its first argument, and $\tilde{U}$ denotes the vector of expansion coefficients. The advantage of the discretization here is that the nonlinear functional equation $R(v^h; u^h) = 0$ has been turned into a nonlinear vector equation $r(\tilde{U}) = 0$, which is possible to solve using computer methods. For a more complete description of finite element spatial discretizations see **?**.

3.1.4 The algorithm

If we combine the space and time discretization schemes described above we obtain a discrete equation for the state of the system at each time level.

The fully discretized problem then reads, find the sequence of vectors $\{\tilde{U}^n\}$ such that

$$0 = r^n(\tilde{U}^n) = \sum_{i,j=1}^{N} \frac{\tilde{U}_i^n - \tilde{U}_i^{n-1}}{k_n} \langle \phi_j,\ \phi_i \rangle + a^t \left(\phi_j; \frac{\tilde{U}_i^n + \tilde{U}_i^{n-1}}{2} \phi_i \right) - b^n, \tag{3.15}$$

holds for all $n \in [1...N_t]$, where N_t is the number of time steps. Here $r^n(\tilde{U}^n) = r(\tilde{U}^{n-1}, \tilde{U}^n)$, and the vector b^n is given in component form by $b_i^n = \left\langle \phi_i,\ \frac{f^n + f^{n-1}}{2} \right\rangle$. It is usual to write $\langle \phi_j,\ \phi_i \rangle = M_{i,j}$, where the matrix M is called the *mass matrix*. With the use of Newton's method the nonlinear operator a^t is linearized, leading to a series of linear systems of equations which can be solved using numerical linear algebra.

3.2 Newton's Method

Newton's method is a technique for approximating the roots of equations using successive linearizations. In Europe the technique was first developed for polynomials around the 17th century by Sir Isaac Newton and Joseph Raphson, and has since then become widely accepted due to its simplicity and wide variety of applications.

In the context of our fully discretized model problem (3.15), we are interested in the vector form of Newton's method. The goal here is to approximate the solution of the equation $r^n(\tilde{U}^n) = 0$. This is done in Algorithm 1.

Algorithm 1: Vector Newton's Method

Choose $\tilde{U}^{n,0}$;
while $\|r^n(\tilde{U}^{n,k})\| > \ TOL$ **do**
    Solve $r'^{n,k} \Delta\tilde{U}^{n,k} = -r^{n,k}$;
    Update $\tilde{U}^{n,k+1} = \Delta\tilde{U}^{n,k} + \tilde{U}^{n,k}$;
end
return $\tilde{U}^{n,k+1}$

In algorithm 1, the vector $\Delta\tilde{U}^{n,k} = \tilde{U}^{n,k+1} - \tilde{U}^{n,k}$, TOL is a given tolerance, and k denotes the Newton iteration number. Note the dependencies of the Jacobian matrix $r'^{n,k} = r'(\tilde{U}^{n-1}, \tilde{U}^{n,k})$ and the residual vector $r^{n,k} = r(\tilde{U}^{n-1}, \tilde{U}^{n,k})$. The vector $\tilde{U}^{n,0}$ is an initial guess which is needed to get Newton's method started. When the algorithm convergences one obtains an approximation of $\tilde{U}^n$ which is accurate up to a residual error of TOL .

One of the key feature's of Newton's method is the so called *quadratic convergence*. If the previous Newton iterate is 'close enough' to the zero, then the residual error in the next iterate will be bound by a multiple of the square of the previous residual error. This result is proved in Kantoroviche's

theorem, which was first stated by the Soviet mathematician Leonid Kantorovich [Kan, 1948].

3.3 Time Discretization of the FSI Equations

Now that the general methods for the discretization of partial differential equations in time and space have been presented, we are ready to discretize the FSI problem in weak form (2.59). We begin here by formulating a time stepping scheme. In doing this it is useful to divide the variables into two groups. These are the *kinematic variables*, $U_{\mathcal{K}} = (U_{_F}, D_{_S}, U_{_S}, D_{_F})$, which represent displacements and velocities, and the *enforcement variables*, $U_{\mathcal{E}} = (P_{_F}, L_{_U}, L_{_D})$, which are used to weakly enforce the conditions of incompressibility, kinematic continuity and domain continuity.

The kinematic variables are time discretized using a standard cG(1) time-stepping scheme which was previously presented in 3.1.3. The kinematic variables $U_{\mathcal{K}}$ are approximated by functions which are linear in time, thereby allowing the exact evaluation of the time derivative terms

$$\int_{t_{n-1}}^{t_n} \dot{U}_{\mathcal{K}} \, \mathrm{d}t = \frac{U_{\mathcal{K}}^n - U_{\mathcal{K}}^{n-1}}{k_n}. \tag{3.16}$$

Here the superscript n denotes the value of the function at the n^{th} time level $U_{\mathcal{K}}^n = U_{\mathcal{K}}(t = t_n)$, and k_n the value of the time step $k_n = t_n - t_{n-1}$. The value of $U_{\mathcal{K}}$ without the time derivative is approximated by the midpoint rule

$$U_{\mathcal{K}} \approx \frac{(U_{\mathcal{K}}^n + U_{\mathcal{K}}^{n-1})}{2}. \tag{3.17}$$

This approximation is exact for terms where the integration variables appears linearly, but inexact for products of variables.

In contrast to the kinematic variables, the enforcement variables are time approximated by a simple endpoint rule $U_{\mathcal{E}} \approx U_{\mathcal{E}}^n$. This is because we want to enforce the conditions that they represent at each time level individually, and not 'on average over time' as would be the effect of a midpoint approximation.

One modification to the above mentioned scheme is made, namely the endpoint approximation of the fluid domain displacement $D_{_F}$ is used in the weak forms of the fluid equations (2.42), instead of the expected midpoint approximation. It was noticed during the running of the blood vessel problem described in Section 5.4, that this modification was more robust, i.e. the numerical scheme with endpoint approximation of $U_{_F}$ in the fluid equation succeeded where the midpoint approximation broke down.

The time discretized version of the weak FSI problem can now be formulated, using the superscript $n - \frac{1}{2}$ to indicate a midpoint approximation. The time discretized weak FSI problem reads, given a partition of the time

interval $\bigcup I_n = [0,T], I_n = (t_{n-1}, t_n]$, find the sequence of functions $\{U^{k,n}\}$ such that the following holds for all n

$$
\begin{aligned}
R^n = &\left\langle v_{_F},\ \rho_{_F} J^n_{_F} \left(\frac{U^n_{_F} - U^{n-1}_{_F}}{k} \right) \right\rangle_{_F} - \left\langle v_{_F},\ J_{_F} b^{n-\frac{1}{2}}_{_F} \right\rangle_{_F} \\
&+ \left\langle v_{_F},\ \operatorname{Grad} U^{n-\frac{1}{2}}_{_F} F^{-1,n}_{_F} \cdot \left(U^{n-\frac{1}{2}}_{_F} - \frac{D^n_{_F} - D^{n-1}_{_F}}{k} \right) \right\rangle_{_F} \\
&+ \left\langle q_{_F},\ \operatorname{Div}\ \left(J^n_{_F} F^{-1,n}_{_F} \cdot U^n_{_F} \right) \right\rangle_{_F} \\
&+ \left\langle \operatorname{Grad} v_{_F},\ J^n_{_F}\, \Sigma_{_F} \left(U^{n-\frac{1}{2}}_{_F}, P^n_{_F}, D^n_{_F} \right) F^{-\top,n}_{_F} \right\rangle_{_F} \\
&+ \langle v_{_F}, \mu_{_F} (F^{-\top,n-\frac{1}{2}}_{_F} \cdot \operatorname{Grad} U^{\top,n-\frac{1}{2}}_{_F}) - P^n_{_F} I \rangle_{\Gamma_{DN}} \\
&+ \left\langle c_{_S},\ \rho_{_S}\ \left(\frac{U^n_{_S} - U^{n-1}_{_S}}{k} \right) \right\rangle_{_S} + \left\langle \operatorname{Grad} v_{_S},\ \Sigma_{_S} \left(D^{n-\frac{1}{2}}_{_S} \right) \right\rangle_{_S} \\
&+ \left\langle v_{_S},\ \frac{D^n_{_S} - D^{n-1}_{_S}}{k} - U^{n-\frac{1}{2}}_{_S} \right\rangle_{_S} + \left\langle c_{_S},\ B^{n-\frac{1}{2}}_{_S} \right\rangle_{_S} \\
&+ \left\langle c_{_F},\ \frac{D^n_{_F} - D^{n-1}_{_F}}{k} \right\rangle_{_F} + \left\langle \operatorname{Grad}^{\mathbf{s}} c_{_F},\ \Sigma_{_{FD}} \left(D^{n-\frac{1}{2}}_{_F} \right) \right\rangle_{_F} \\
&- \left\langle c_{_S},\ J^{n-\frac{1}{2}}_{_F} \Sigma_{_F} \left(U^{n-\frac{1}{2}}_{_F}, P^n_{_F}, D^{n-\frac{1}{2}}_{_F} \right) F^{-\top,n-\frac{1}{2}}_{_F} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}} \\
&+ \left\langle c_{_F},\ L^n_{_D} \right\rangle_{\Gamma_{_{FSI}}} + \left\langle m_{_D},\ D^n_{_F} - D^n_{_S} \right\rangle_{\Gamma_{_{FSI}}} \\
&+ \left\langle v_{_F},\ L^n_{_U} \right\rangle_{\Gamma_{_{FSI}}} + \left\langle m_{_U},\ U^n_{_F} - U^n_{_S} \right\rangle_{\Gamma_{_{FSI}}} \\
&= 0.
\end{aligned}
\tag{3.18}
$$

Here R^n denotes the time discretized residual for time level n. Note that the initial fluid pressure is never used in the fluid part of the time discretized residual, but is rather used in the terms coming from the stress continuity interface condition. This is because the fluid pressure acts as an enforcement variable for the fluid equations, and as a kinematic variable in the interface condition.

3.4 Finite Element Discretization of the FSI Equations

At each time level t_n, (3.18) defines a spatial partial differential equation for the current state of the system, U^n. In order to solve this equation numerically, we apply the general finite element method given in Section 3.1 to our specific problem. This involves the division of the computational domain

Ω into a mesh of simplicial cells $\mathcal{K}_i$ such that $\bigcup \mathcal{K}_i = \Omega$. In the context of this thesis the cells are arranged so that Γ_{FSI} lies completely on element boundaries, and without any hanging vertices on either side of the interface. This restriction simplifies the implementation of the solver considerably, but does not allow different sized elements to be used on opposing sides of the fluid-structure boundary.

The global trial and test functions are now restricted to finite dimensional function spaces, marked by superscripts h, i.e. we seek $(U^{h,n}, v^h) \in (V^h, \hat{V}^h)$. The mixed discrete trial and test spaces are given by

$$V^h = (V_F^h \times Q_F^h \times M_U^h \times C_S^h \times V_S^h \times C_F^h \times M_D^h), \tag{3.19}$$

and

$$\hat{V}^h = (\hat{V}_F^h \times \hat{Q}_F^h \times \hat{M}_U^h \times \hat{C}_S^h \times \hat{V}_S^h \times \hat{C}_F^h \times \hat{M}_D^h). \tag{3.20}$$

The various subspaces of V_h and $\hat{V}_h$ are Lagrange spaces with polynomial basis function degree q, denoted cG(q) or discontinuous spaces with polynomial basis function degree q, denoted dG(q). Two choices of mixed function spaces are considered in this thesis, those corresponding to $p = 1$ and $p = 2$ in Table 3.1.

Table 3.1: Function spaces used for the finite element spatial discretization of the total residual equation (3.18).

V_F^h	Q_F^h	M_U^h	C_S^h	V_S^h	C_F^h	M_D^h
cG(2)	cG(1)	dG(0)	cG(p)	cG(p)	cG(p)	dG(0)

In both cases, the fluid spaces use a Taylor–Hood type discretization [Taylor and Hood, 1973], with piecewise constant elements for the Lagrange multipliers. It was noticed during simulation that this choice for the Lagrange multiplier spaces results in a 'smoother' pressure in the corners of the fluid domain that touch Γ_{FSI}. The difference between the two choices of space discretization lie in different degrees for the function spaces corresponding to the variables (D_S, U_S, D_F). Simulations with $p = 1$ are cheaper to compute, whereas simulations with $p = 2$ have a better match of displacements and velocities across Γ_{FSI}.

We can now formulate the fully discrete FSI problem by setting in the discrete functions into the time discretized problem (3.18), this yields

$$R^n(v^h; U^{h,n}) = 0 \quad \forall n. \tag{3.21}$$

The trial function $U^{h,n}$ can be written as a sum of basis functions

$$U^{h,n} = \sum_{i=1}^{N} \tilde{U}_i^n \phi_i(X) \tag{3.22}$$

where $\tilde{U}_i^n$ are the expansion coefficients, $\phi_i(X)$ the i^{th} basis function, and N the dimension of the function space V^h. The test function v^h can also be expanded into a linear combination of basis functions. Since v^h appears linearly in R^n, the expansion coefficients for v^h can be divided out of equation 3.21 to yield

$$R^n\left(\phi_j; \sum_{i=1}^{N} \tilde{U}_i^n \phi_i\right) = 0. \tag{3.23}$$

Solving this equation is equivalent to finding a zero in the vector function $r^n : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, whose j^{th} component is given by

$$r^n(\tilde{U})_j = R^n\left(\phi_j; \sum_{i=1}^{N} \tilde{U}_i^n \phi_i\right). \tag{3.24}$$

This means that the state of the FSI system at time t_n can be determined either by finding a zero of the nonlinear functional R^n, or equivalently finding a zero of the vector function r^n.

3.5 Newton's Method for FSI

In the case of the FSI problem, a Newton's method for the solution of $r^n = 0$ at every time level can be obtained simply by applying Algorithm 1. However, the implementation of the assembly of $r'^{n,k}$ and $r^{n,k}$ from scratch would be a very daunting task, requiring the handling of integration over arbitrary simplicies of products of finite element basis functions to a sufficiently high degree, as well as a large amount of bookkeeping needed to properly track the many components of tensor products. Thankfully these tasks have been automated by the software of the FEniCS project, allowing us to work on a more abstract level.

As a functional equation, the solve part of algorithm 1 reads

$$R'^{n,k}[\Delta U^{n,k}] = -R^{n,k}. \tag{3.25}$$

Here $\Delta U^{n,k} = U^{n,k+1} - U^{n,k}$, where $U^{n,k}$ denotes the k^{th} Newton iterate function at time level t_n, $R^{n,k} = R^n(U^{n,k})$, and $R'^{n,k}$ denotes the Fréchet derivative of $R^{n,k}$ with respect to $U^{n,k}$. It is assumed that $R^{n,k}$ is Fréchet differentiable for all n and k.

Using FEniCS we can specify R'^n and R^n using Unified Form Language, and then automatically assemble $r'^{n,k}$ and $r^{n,k}$. The linear system $r'^{n,k}\Delta\tilde{U}^{n,k} = -r^{n,k}$ from the vector Newton's method can then be solved using numerical linear algebra.

For our FSI problem R^n is given by (3.18). The Fréchet derivative R'^n can be calculated manually (see appendix 3.6), or it can be derived from R^n using symbolic differentiation in FEniCS.

In the implementation, the initial guess in Newton's method has simply been chosen as the value of the system state at the previous time level. A possible improvement might be to extrapolate the initial guess from the previous system states, which might secure faster convergence.

3.6 Calculation of the Fréchet derivative R'^n

In this section, we present details of the calculation of the Fréchet derivative R'^n as an important step in the derivation of the Newton's method solution algorithm. The calculations here were adapted from those done in Selim [2011]. We consider at first the spatial residual $\tilde{R}$ such that $\int_0^T \tilde{R}\ \mathrm{d}t = R$, with R given by equation 2.59. The derivative of this residual is broken up into fluid, structure, fluid domain and interface derivatives by

$$\tilde{R}' = \tilde{R}'_F + \tilde{R}'_S + \tilde{R}'_{FD} + \tilde{R}'_{\Gamma_{FSI}} \tag{3.26}$$

3.6.1 Preliminaries

We recall that the functional derivative $D_{\delta v}[\ \mathcal{F}\ ](v)$ (Gâteaux derivative) of an operator $\mathcal{F} : V \to W$ in a direction $\delta v \in V$ at a point $v \in V$ is defined as

$$D_{\delta v}[\ \mathcal{F}\ ](v) = \lim_{\epsilon \to 0} \frac{\mathcal{F}(v + \epsilon \delta v) - \mathcal{F}(v)}{\epsilon}. \tag{3.27}$$

We usually omit the argument v and write $D_{\delta v}[\ \mathcal{F}\ ](v) = D_{\delta v}[\ \mathcal{F}\ ]$. It can be shown that an operator is Gâteaux differentiable in all directions if it is Fréchet differentiable. We will now make use of the following rules:

Product Rule

Let F, G be tensor valued functions. The derivative of the L^2 inner product of the functions is given by

$$D_{\delta v}[\ \langle F,\ G\rangle\ ] = \langle D_{\delta v}[\ F\ ],\ G\rangle + \langle F,\ D_{\delta v}[\ G\ ]\rangle\,. \tag{3.28}$$

The derivative of an inverse

Let F be an invertible matrix-valued operator. The functional derivative of F^{-1} is then given by

$$D_{\delta v}[\ F^{-1}\ ] = -F^{-1} D_{\delta v}[\ F\ ] F^{-1}. \tag{3.29}$$

This follows by considering the derivative of $I = FF^{-1}$. We similarly find that

$$D_{\delta v}[\ F^{-\top}\ ] = -F^{-\top} D_{\delta v}[\ F^{\top}\ ] F^{-\top}. \tag{3.30}$$

In particular, if $F = I + \text{Grad}\, v$, then

$$D_{\delta v}[\, F^{-1}\,] = -F^{-1}\text{Grad}\,\delta v F^{-1}, \tag{3.31}$$
$$D_{\delta v}[\, F^{-\top}\,] = -F^{-\top}(\text{Grad}\,\delta v)^{\top}F^{-\top}. \tag{3.32}$$

The derivative of a determinant

Let J be the determinant of an invertible matrix-valued operator F. The functional derivative of J is given by

$$D_{\delta v}[\, J\,] = J\ \text{tr}(D_{\delta v}[\, F\,]F^{-1}). \tag{3.33}$$

See Gurtin [1981] for a proof. In particular, if $F = I + \text{Grad}\, v$, then

$$D_{\delta v}[\, J\,] = J\ \text{tr}(\text{Grad}\,\delta v F^{-1}). \tag{3.34}$$

3.6.2 Linearization of the Fluid Residual $\tilde{R}_F$

We differentiate the fluid residual $\tilde{R}_F$ with respect to the variables $(U_{_F}, P_{_F}, D_{_F})$. We then need to differentiate the following terms

$$\text{D}_{_F}^{(t)} = \Big\langle v_{_F},\ \rho_{_F} J_{_F}(\dot{U}_{_F} + \text{Grad}\, U_{_F}F_{_F}^{-1}\cdot(U_{_F} - \dot{D}_{_F}))\Big\rangle_{F}, \tag{3.35}$$
$$\widetilde{\Sigma}_{_F} = \big\langle v_{_F},\ J_{_F}\mu_{_F}\text{Grad}\, U_{_F}F_{_F}^{-1}F_{_F}^{-\top}\big\rangle_{F},$$
$$+\ \big\langle v_{_F},\ J_{_F}(\mu_{_F}(F_{_F}^{-\top}\text{Grad}\, U_{_F}^{\top}) - P_{_F}I)F_{_F}^{-\top}\big\rangle_{F} \tag{3.36}$$
$$\text{Div}_{_F} = \big\langle q_{_F},\ \text{Div}\,(J_{_F}F_{_F}^{-1}U_{_F})\big\rangle_{F}, \tag{3.37}$$
$$-G_{\Gamma_{DN}} = \big\langle v_{_F},\ -J_{_F}(\mu_{_F}F_{_F}^{\top}\text{Grad}\, U_{_F}^{\top} - P_{_F}I)F_{_F}^{-\top}\cdot N_{_F}\big\rangle_{\Gamma_{DN}}, \tag{3.38}$$
$$-B_{_F} = -\,\big\langle v_{_F},\ J_{_F}b_{_F}\big\rangle_{F}. \tag{3.39}$$

$D_{\delta U_F}[\, \tilde{R}_F\,]$

We find that

$$D_{\delta U_F}[\,\text{D}_{_F}^{(t)}\,] = \Big\langle v_{_F},\ \rho_{_F}J_{_F}(\delta\dot{U}_{_F} + \text{Grad}\,\delta U_{_F}F_{_F}^{-1}\cdot(U_{_F} - \dot{D}_{_F})\Big\rangle_{F}$$
$$+\ \big\langle v_{_F},\ \text{Grad}\, U_{_F}F_{_F}^{-1}\cdot\delta U_{_F})\big\rangle_{F},$$
$$D_{\delta U_F}[\,\widetilde{\Sigma}_{_F}\,] = \big\langle v_{_F},\ J_{_F}\mu_{_F}(\text{Grad}\,\delta U_{_F}F_{_F}^{-1} + F_{_F}^{-\top}\text{Grad}\,\delta U_{_F}^{\top})F_{_F}^{-\top}\big\rangle_{F},$$
$$D_{\delta U_F}[\,\text{Div}_{_F}\,] = \big\langle q_{_F},\ \text{Div}\,(J_{_F}F_{_F}^{-1}\delta U_{_F})\big\rangle_{F},$$
$$D_{\delta U_F}[\,-G_{\Gamma_{DN}}\,] = -\big\langle v_{_F},\ J_{_F}\mu_{_F}F_{_F}^{-\top}\text{Grad}\,\delta U_{_F}^{\top}F_{_F}^{-\top}\cdot N_{_F}\big\rangle_{\Gamma_{DN}},$$
$$D_{\delta U_F}[\,-B_{_F}\,] = 0.$$

$D_{\delta P_F}[\, \tilde{R}_F\,]$

Differentiating with respect to the fluid pressure yields

$$D_{\delta P_F}[\,\widetilde{\Sigma}_{_F}\,] = -\big\langle v_{_F},\ J_{_F}\delta P_{_F}IF_{_F}^{-\top}\big\rangle_{F},$$
$$D_{\delta P_F}[\,-G_{\Gamma_{DN}}\,] = \big\langle v_{_F},\ J_{_F}\delta P_{_F}IF_{_F}^{-\top}\cdot N_{_F}\big\rangle_{\Gamma_{DN}}$$
$$D_{\delta P_F}[\,\text{Div}_{_F}\,] = D_{\delta P_F}[\,\text{D}_{_F}^{(t)}\,] = D_{\delta P_F}[\,-B_{_F}\,] = 0.$$

$D_{\delta D_F}[\,\tilde{R}_F\,]$

Using (3.34), we find that

$$\begin{aligned}
D_{\delta D_F}[\,\mathrm{D}_F^{(t)}\,] &= \Big\langle v_F,\ \rho_F J_F \mathrm{tr}(\mathrm{Grad}\,\delta U_F F_F^{-1})(\dot{U}_F + \mathrm{Grad}\,U_F F_F^{-1}\cdot(U_F - \dot{D}_F))\Big\rangle_F\,,\\
&- \Big\langle v_F,\ \rho_F J_F \mathrm{Grad}\,U_F F_F^{-1}(\mathrm{Grad}\,\delta D_F F_F^{-1}\cdot(U_F - \dot{D}_F) + \delta\dot{D}_F)\Big\rangle_F\,,\\
D_{\delta D_F}[\,\widetilde{\Sigma}_F\,] &= \big\langle v_F,\ J_F \mathrm{tr}(\mathrm{Grad}\,\delta D_F F_F^{-1})\Sigma_F F_F^{-\top}\big\rangle_F\,,\\
&- \big\langle v_F,\ J_F(\mu_F \mathrm{Grad}\,U_F F_F^{-1}\mathrm{Grad}\,\delta D_F F_F^{-1})F_F^{-\top}\big\rangle_F\,,\\
&- \big\langle v_F,\ J_F(\mu_F F_F^{-\top}\mathrm{Grad}\,\delta D_F^\top F_F^{-\top}\mathrm{Grad}\,U_F^\top)F_F^{-\top}\big\rangle_F\,,\\
&- \big\langle v_F,\ J_F\Sigma_F F_F^{-\top}\mathrm{Grad}\,\delta D_F^\top F_F^{-\top}\big\rangle_F\,,\\
D_{\delta D_F}[\,\mathrm{Div}_F\,] &= \big\langle q_F,\ \mathrm{Div}\,(J_F(\mathrm{tr}(\mathrm{Grad}\,\delta D_F F_F^{-1})I - F_F^{-1}\mathrm{Grad}\,\delta D_F)F_F^{-1}\cdot U_F)\big\rangle_F\,,\\
D_{\delta D_F}[\,-G_{\Gamma_{DN}}\,] &= \big\langle v_F,\ J_F(\mathrm{tr}(\mathrm{Grad}\,\delta D_F F_F^{-1})\mu_F F_F^{-\top}\mathrm{Grad}\,U_F^\top)F_F^{-\top}\cdot N_F\big\rangle_{\Gamma_{DN}}\,,\\
&- \big\langle v_F,\ J_F(\mu_F F_F^{-\top}\mathrm{Grad}\,\delta D_F^\top F_F^{-\top}\mathrm{Grad}\,U_F^\top)F_F^{-\top}\cdot N_F\big\rangle_{\Gamma_{DN}}\,,\\
&- \big\langle v_F,\ J_F(\mu_F F_F^{-\top}\mathrm{Grad}\,U_F^\top)F_F^{-\top}\mathrm{Grad}\,\delta D_F^\top F_F^{-\top}\cdot N_F\big\rangle_{\Gamma_{DN}}\,,\\
D_{\delta D_F}[\,-B_F\,] &= -\big\langle v_F,\ J_F\ \mathrm{tr}(\mathrm{Grad}\,\delta D_F F_F^{-1})b_F\big\rangle_F\,.
\end{aligned}$$

Note that the untransformed fluid tensor Σ_F is here given by

$$\Sigma_F(U_F, P_F, D_F) = \mu_F\left(\mathrm{Grad}\,U_F F_F^{-1} + F_F^{-\top}\mathrm{Grad}\,U_F^\top\right) - P_F I. \tag{3.40}$$

3.6.3 Linearization of the Structure Residual $\tilde{R}_S$

We differentiate the structure residual $\tilde{R}_S$ with respect to its arguments (D_S, U_S). Here the following terms are considered,

$$\begin{aligned}
\mathrm{D}_S^{(t)} &= \Big\langle v_S,\ \dot{D}_S - U_S\Big\rangle_S\,,\\
\mathrm{D}_S^{(tt)} &= \Big\langle c_S,\ \rho_S\dot{U}_S\Big\rangle_S\,,\\
\Sigma_S &= \langle c_S,\ F_S(2\mu_S E_S + \lambda_S \mathrm{tr}(E_S)I)\rangle_S\,.
\end{aligned}$$

As a reminder, $E_S = \frac{1}{2}(F_S^\top F_S - I)$ and $F_S = I + \mathrm{Grad}\,D_S$. In order to simplify the calculations we precalculate the derivative of E_S.

$$D_{\delta D_S}[\,E_S\,] = \tfrac{1}{2}(\mathrm{Grad}\,\delta D_S^\top(I + \mathrm{Grad}\,D_S) + (I + \mathrm{Grad}\,D_S^\top)\mathrm{Grad}\,\delta D_S).$$

$D_{\delta D_S}[\,\tilde{R}_S\,]$

Differentiating $\tilde{R}_S$ in the direction D_S, we obtain

$$\begin{aligned}
D_{\delta D_S}[\,\mathrm{D}_S^{(t)};]\,] &= \Big\langle v_S,\ \delta\dot{D}_S\Big\rangle_S\,,\\
D_{\delta D_S}[\,\mathrm{D}_S^{(tt)};]\,] &= 0,\\
D_{\delta D_S}[\,\Sigma_S\,] &= \langle c_S,\ \mathrm{Grad}\,\delta D_S(2\mu_S E_S + \lambda_S \mathrm{tr}(E_S)I)\rangle_S\\
&+ \langle c_S,\ F_S(2\mu_S D_{\delta D_S}[\,E_S\,] + \lambda_S \mathrm{tr}(D_{\delta D_S}[\,E_S\,])I)\rangle_S\,.
\end{aligned}$$

$D_{\delta U_S}[\,\tilde{R}_S\,]$

Differentiating in the direction U_S, we see that

$$\begin{aligned} D_{\delta U_S}[\,\mathrm{D}_S^{(tt)}\,] &= \left\langle c_S,\ \rho_S \delta\dot{U}_S \right\rangle_S, \\ D_{\delta U_S}[\,\mathrm{D}_S^{(t)}\,] &= -\left\langle v_S,\ \delta U_S \right\rangle, \\ D_{\delta U_S}[\,\Sigma_S\,] &= 0. \end{aligned}$$

3.6.4 Linearization of the Fluid Domain Residual $\tilde{R}_{FD}$

We differentiate the fluid domain residual $\tilde{R}_{FD}$ with respect to D_F. We consider the terms of the residual

$$\begin{aligned} \mathrm{D}_{FD}^{(t)} &= \left\langle c_F,\ \delta\dot{D}_F \right\rangle_F, \\ \Sigma_{FD} &= \left\langle c_F,\ 2\mu_{FD}\mathrm{Grad}^{\mathbf{s}}D_F + \lambda_{FD}\mathrm{tr}(\mathrm{Grad}\,D_F)I \right\rangle_F. \end{aligned} \tag{3.41}$$

$D_{\delta D_F}[\,\tilde{R}_{FD}\,]$

As $\tilde{R}_{FD}$ is completely linear, the differentiation can be accomplished by simply substituting D_F with δD_F, yielding

$$\begin{aligned} D_{\delta D_F}[\,\mathrm{D}_{FD}^{(t)}\,] &= \left\langle c_F,\ \delta\dot{D}_F \right\rangle_F, \\ D_{\delta D_F}[\,\Sigma_{FD}\,] &= \left\langle c_F,\ 2\mu_{FD}\mathrm{Grad}^{\mathbf{s}}\delta D_F + \lambda_{FD}\mathrm{tr}(\mathrm{Grad}\,\delta D_F)I \right\rangle_F. \end{aligned}$$

3.6.5 Linearization of the Interface Residual $\tilde{R}_{\Gamma_{FSI}}$

This residual can be split up into linear and nonlinear parts $\tilde{R}_{\Gamma_{FSI}} = \tilde{R}^l_{\Gamma_{FSI}} + \tilde{R}^{nl}_{\Gamma_{FSI}}$, which are given in functional form by,

$$\begin{aligned} \tilde{R}^l_{\Gamma_{FSI}} &= \left\langle v_F,\ L_U \right\rangle_{\Gamma_{FSI}} + \left\langle m_U,\ U_F - U_S \right\rangle_{\Gamma_{FSI}}, \\ &\quad + \left\langle c_F,\ L_D \right\rangle_{\Gamma_{FSI}} + \left\langle m_D,\ D_F - D_S \right\rangle_{\Gamma_{FSI}}, \\ \tilde{R}^{nl}_{\Gamma_{FSI}} &= \left\langle c_S,\ \widetilde{\Sigma}_F \cdot N_F \right\rangle_{\Gamma_{FSI}}. \end{aligned}$$

$\tilde{R}^{l'}_{\Gamma_{FSI}}$

The Fréchet derivative of the linear operator $\tilde{R}^l_{\Gamma_{FSI}}$ is obtained by simply substituting trial variables with their differentials. This yields

$$\begin{aligned} \tilde{R}^{l'}_{\Gamma_{FSI}} &= \left\langle v_F,\ \delta L_U \right\rangle_{\Gamma_{FSI}} + \left\langle m_U,\ \delta U_F - \delta U_S \right\rangle_{\Gamma_{FSI}}, \\ &\quad + \left\langle c_F,\ \delta L_D \right\rangle_{\Gamma_{FSI}} + \left\langle m_D,\ \delta D_F - \delta D_S \right\rangle_{\Gamma_{FSI}}. \end{aligned}$$

$\tilde{R}^{nl'}_{\Gamma_{FSI}}$

The nonlinear term $\tilde{R}^{nl}_{\Gamma_{FSI}}$ is then differentiated in the relevant directions (U_F, P_F, D_F). Here the results from the derivation of $\widetilde{\Sigma}_F$ in section 3.6.2 can be reused. In summary one obtains, with a slight abuse of notation,

$$\begin{aligned}\tilde{R}^{nl'}_{\Gamma_{FSI}} &= \left\langle v_S,\ D_{\delta U_F}[\,\widetilde{\Sigma}_F\,]\cdot N_F \right\rangle_{\Gamma_{FSI}} + \left\langle v_S,\ D_{\delta P_F}[\,\widetilde{\Sigma}_F\,]\cdot N_F \right\rangle_{\Gamma_{FSI}}, \\ &\quad + \left\langle v_S,\ D_{\delta D_F}[\,\widetilde{\Sigma}_F\,]\cdot N_F \right\rangle_{\Gamma_{FSI}}.\end{aligned}$$

with the formula being correct if we ignore the integral and test function parts of $D_{\delta U_F}[\,\widetilde{\Sigma}_F\,], D_{\delta P_F}[\,\widetilde{\Sigma}_F\,]$ and $D_{\delta D_F}[\,\widetilde{\Sigma}_F\,]$.

3.6.6 The Time Discretized Derivative R'^n

Using the above calculations for the computation of the Frechét derivatives, $(\tilde{R}'_F, \tilde{R}'_S, \tilde{R}'_{FD}, \tilde{R}'_{\Gamma_{FSI}})$, we can now formulate the time discretized derivative R'^n. To do this we make use of the following notation, which attempts to mimic the way that $\tilde{R}$ can be derived from $(\tilde{R}_F, \tilde{R}_S, \tilde{R}_{FD}, \tilde{R}_{\Gamma_{FSI}})$ by making substitutions according to the time stepping scheme used in section 3.3.

$$\begin{array}{llll}
\dot{U}^n_F = \frac{U^n_F - U^{n-1}_F}{k} & \dot{D}^n_S = \frac{D^n_S - D^{n-1}_S}{k} & \dot{U}^n_S = \frac{U^n_S - U^{n-1}_S}{k} & \dot{D}^n_F = \frac{D^n_F - D^{n-1}_F}{k} \\
\delta\dot{U}^n_F = \frac{\delta U^n_F}{k} & \delta\dot{D}^n_S = \frac{\delta D^n_S}{k} & \delta\dot{U}^n_S = \frac{\delta U^n_S}{k} & \delta\dot{D}^n_F = \frac{D^n_F}{k} \\
U^{n-\frac{1}{2}}_F = \frac{U^n_F + U^{n-1}_F}{2} & D^{n-\frac{1}{2}}_S = \frac{D^n_S + D^{n-1}_S}{2} & & D^{n-\frac{1}{2}}_F = \frac{D^n_F + D^{n-1}_F}{2} \\
\delta U^{n-\frac{1}{2}}_F = \frac{\delta U_F}{2} & \delta D^{n-\frac{1}{2}}_S = \frac{\delta D_S}{2} & & \delta D^{n-\frac{1}{2}}_F = \frac{\delta D_F}{2}
\end{array} \tag{3.42}$$

Now let the trial functions, test functions, and Frechét differentials be defined as

$$\begin{aligned}
U^n &= (U^n_F, P^n_F, L^n_U, D^n_S, U^n_S, D^n_F, L^n_D) && \in V, \\
U^{n-1} &= (U^{n-1}_F, P^{n-1}_F, L^{n-1}_U, D^{n-1}_S, U^{n-1}_S, D^{n-1}_F, L^{n-1}_D) && \in V, \\
v &= (v_F, q_F, m_U, c_S, v_S, c_F, m_D) && \in \hat{V}, \\
\delta U &= (\delta U_F, \delta P_F, \delta L_U, \delta D_S, \delta U_S, \delta D_F, \delta L_D) && \in V.
\end{aligned} \tag{3.43}$$

where $V = (V_F \times Q_F \times M_U \times C_S \times V_S \times C_F \times M_D)$ is the global trial function space, and $\hat{V} = (\hat{V}_F \times \hat{Q}_F \times \hat{M}_U \times \hat{C}_S \times \hat{V}_S \times \hat{C}_F \times \hat{M}_D)$ the global test function space.

The Fréchet derivative $R'^n(v, \delta U; U)$ of the residual R^n(equation 3.18) is given by:

$$
\begin{aligned}
&R'^n(v, \delta U; U^n, U^{n-1}) = \\
&\\
&\Big\langle v_{_F},\ \rho_{_F} J^n_{_F}(\delta \dot{U}^n_{_F} + \text{Grad}\ \delta U^{n-\frac{1}{2}}_{_F} F^{-1,n}_{_F} \cdot (U^{n-\frac{1}{2}}_{_F} - \dot{D}^n_{_F}) + \text{Grad}\, U^{n-\frac{1}{2}}_{_F} F F^{-1,n} \cdot \delta U^{n-\frac{1}{2}}_{_F}\Big\rangle_{_F}, \\
&+ \Big\langle \text{Grad}\, v_{_F},\ J^n_{_F}\mu_{_F}(\text{Grad}\ \delta U^{n-\frac{1}{2}}_{_F} F^{-1,n}_{_F} + F^{-\top,n}_{_F} \text{Grad}\ \delta U^{\top,n}_{_F}) F^{-\top,n}_{_F}\Big\rangle_{_F}, \\
&- \big\langle \text{Grad}\, v_{_F},\ J^n_{_F} F^{-1,n}_{_F} \delta P^n_{_F}\big\rangle_{_F} + \Big\langle q_{_F},\ \ \text{Div}\ (J^n_{_F} F^{-1,n}_{_F} \cdot \delta U^{n-\frac{1}{2}}_{_F})\Big\rangle_{_F}, \\
&- \big\langle v_{_F},\ \ J^n_{_F}\ \text{tr}(\text{Grad}\ \delta D^n_{_F} F^{-1,n}_{_F}) b_{_F}\big\rangle_{_F}, \\
&- \big\langle v_{_F},\ J_{_F}(\mu_{_F} F^{-\top}_{_F} \text{Grad}\ \delta U^{\top}_{_F} - \delta P_{_F} I) F^{-\top}_{_F} \cdot N_{_F}\big\rangle_{\Gamma_{\Gamma_{DN}}} + \big\langle v_{_F},\ J_{_F} \delta P_{_F} I F^{-\top}_{_F} \cdot N_{_F}\big\rangle_{\Gamma_{\Gamma_{DN}}}, \\
&+ \big\langle v_{_F},\ J_{_F}(\text{tr}(\text{Grad}\ \delta D_{_F} F^{-1}_{_F}) \mu_{_F} F^{-\top}_{_F} \text{Grad}\, U^{\top}_{_F}) F^{-\top}_{_F} \cdot N_{_F}\big\rangle_{\Gamma_{\Gamma_{DN}}}, \\
&- \big\langle v_{_F},\ J_{_F}(\mu_{_F} F^{-\top}_{_F} \text{Grad}\ \delta D^{\top}_{_F} F^{-\top}_{_F} \text{Grad}\, U^{\top}_{_F}) F^{-\top}_{_F} \cdot N_{_F}\big\rangle_{\Gamma_{\Gamma_{DN}}}, \\
&- \big\langle v_{_F},\ J_{_F}(\mu_{_F} F^{-\top}_{_F} \text{Grad}\, U^{\top}_{_F}) F^{-\top}_{_F} \text{Grad}\ \delta D^{\top}_{_F} F^{-\top}_{_F} \cdot N_{_F}\big\rangle_{\Gamma_{\Gamma_{DN}}}, \\
&+ \Big\langle v_{_F},\ \ \rho_{_F} J^n_{_F} \text{tr}(\text{Grad}\ \delta D^n_{_F} F^{-1}_{_F})(\dot{U}^n_{_F} + \text{Grad}\, U^{n-\frac{1}{2}}_{_F} F^{-1,n}_{_F}(U^{n-\frac{1}{2}}_{_F} - \dot{D}^n_{_F}))\Big\rangle_{_F}, \\
&- \Big\langle v_{_F},\ \ \rho_{_F} J^n_{_F} \text{Grad}\, U^{n-\frac{1}{2}}_{_F} F^{-1,n}_{_F}(\text{Grad}\ \delta D^n_{_F} F^{-1,n}_{_F}(U^{n-\frac{1}{2}}_{_F} - \dot{D}^n_{_F}) + \delta \dot{D}^n_{_F})\Big\rangle_{_F}, \\
&+ \big\langle \text{Grad}\, v_{_F},\ \ J^n_{_F} \text{tr}(\text{Grad}\ \delta D^n_{_F} F^{-1,n}_{_F}) \Sigma_{_F} F^{-\top,n}_{_F}\big\rangle_{_F}, \\
&- \Big\langle \text{Grad}\, v_{_F},\ \ J^n_{_F}(\mu_{_F} \text{Grad}\, U^{n-\frac{1}{2}}_{_F} F^{-1}_{_F} \text{Grad}\ \delta D^n_{_F} F^{-1,n}_{_F}) F^{-\top,n}_{_F}\Big\rangle_{_F}, \\
&- \Big\langle \text{Grad}\, v_{_F},\ \ J^n_{_F}(\mu_{_F} F^{-\top,n}_{_F} \text{Grad}\ \delta D^{\top,n}_{_F} F^{-\top,n}_{_F} \text{Grad}\, U^{\top,n-\frac{1}{2}}_{_F}) F^{-\top,n}_{_F}\Big\rangle_{_F}, \\
&- \Big\langle \text{Grad}\, v_{_F},\ \ J^n_{_F} \Sigma_{_F}(U^{n-\frac{1}{2}}_{_F}, P^n_{_F}, D^n_{_F}) F^{-\top,n}_{_F} \text{Grad}\ \delta D^{\top,n}_{_F} F^{-\top,n}_{_F}\Big\rangle_{_F}, \\
&+ \Big\langle q_{_F},\ \ \text{Div}\ (J^n_{_F}(\text{tr}(\text{Grad}\ \delta D^n_{_F} F^{-1,n}_{_F}) I - F^{-1,n}_{_F} \text{Grad}\ \delta D^n_{_F}) F^{-1,n}_{_F} \cdot U^{n-\frac{1}{2}}_{_F})\Big\rangle_{_F}, \\
&+ \Big\langle c_{_S},\ \ \rho_{_S} \delta \dot{U}^n_{_S}\Big\rangle_{_S} + \Big\langle v_{_S},\ \delta \dot{D}^n_{_S} - \delta U^{n-\frac{1}{2}}_{_S}\Big\rangle_{_S}, \\
&+ \Big\langle \text{Grad}\, c_{_S},\ \text{Grad}\ \delta D^{n-\frac{1}{2}}_{_S}(2\mu_{_S} E^{n-\frac{1}{2}}_{_S} + \lambda_{_S} \text{tr}(E^{n-\frac{1}{2}}_{_S}) I)\Big\rangle_{_S}, \\
&+ \Big\langle \text{Grad}\, c_{_S},\ F^{n-\frac{1}{2}}_{_S} \mu_{_S}(\text{Grad}\ \delta D^{\top,n-\frac{1}{2}}_{_S}(I + \text{Grad}\, D^{n-\frac{1}{2}}_{_S}))\Big\rangle_{_S}, \\
&+ \Big\langle \text{Grad}\, c_{_S},\ F^{n-\frac{1}{2}}_{_S} \mu_{_S}(I + \text{Grad}\, D^{\top,n-\frac{1}{2}}_{_S}) \text{Grad}\ \delta D^{n-\frac{1}{2}}_{_S})\Big\rangle_{_S}, \\
&+ \Big\langle \text{Grad}\, c_{_S},\ F^{n-\frac{1}{2}}_{_S} \lambda_{_S} \text{tr}(\tfrac{1}{2}(\text{Grad}\ \delta D^{\top,n-\frac{1}{2}}_{_S}(I + \text{Grad}\, D^{n-\frac{1}{2}}_{_S})))I\Big\rangle_{_S}, \\
&+ \Big\langle \text{Grad}\, c_{_S},\ F^{n-\frac{1}{2}}_{_S} \lambda_{_S} \text{tr}(\tfrac{1}{2}((I + \text{Grad}\, D^{\top,n-\frac{1}{2}}_{_S}) \text{Grad}\ \delta D^{n-\frac{1}{2}}_{_S}))I\Big\rangle_{_S}, \\
&+ \Big\langle c_{_F},\ \delta \dot{D}^n_{_F}\Big\rangle_{_F} + \Big\langle \text{Grad}^{\mathbf{s}} c_{_F},\ \ 2\mu_{_{FD}} \text{Grad}^{\mathbf{s}} \delta D^{n-\frac{1}{2}}_{_F} + \lambda_{_{FD}} \text{tr}(\text{Grad}\ \delta D^{n-\frac{1}{2}}_{_F}) I\Big\rangle_{_F}, \\
&- \Big\langle c_{_S},\ J^{n-\frac{1}{2}}_{_F} \mu_{_F}(\text{Grad}\ \delta U^{n-\frac{1}{2}}_{_F} F^{-1}_{_F} + F^{-\top,n-\frac{1}{2}}_{_F} \text{Grad}\ \delta U^{\top,n-\frac{1}{2}}_{_F}) F^{-\top,n-\frac{1}{2}}_{_F} \cdot N_{_F}\Big\rangle_{\Gamma_{_{FSI}}},
\end{aligned}
$$

$$
\begin{aligned}
&+ \left\langle c_{_S},\ J_{_F}^{n-\frac{1}{2}} \delta P_{_F}^{n} I F_{_F}^{-\top,n-\frac{1}{2}} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}}, \\
&+ \left\langle c_{_S},\ J_{_F}^{n-\frac{1}{2}} (\mathrm{tr}(\mathrm{Grad}\ \delta D_{_F}^{n-\frac{1}{2}} F_{_F}^{-1,n-\frac{1}{2}}) \mu_{_F} F_{_F}^{-\top,n-\frac{1}{2}} \mathrm{Grad}\ U_{_F}^{\top,n-\frac{1}{2}}) F_{_F}^{-\top,n-\frac{1}{2}} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}}, \\
&- \left\langle c_{_S},\ J_{_F}^{n-\frac{1}{2}} (\mu_{_F} F_{_F}^{-\top,n-\frac{1}{2}} \mathrm{Grad}\ \delta D_{_F}^{\top,n-\frac{1}{2}} F_{_F}^{-\top,n-\frac{1}{2}} \mathrm{Grad}\ U_{_F}^{\top,n-\frac{1}{2}}) F_{_F}^{-\top,n-\frac{1}{2}} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}}, \\
&- \left\langle c_{_S},\ J_{_F}^{n-\frac{1}{2}} (\mu_{_F} F_{_F}^{-\top,n-\frac{1}{2}} \mathrm{Grad}\ U_{_F}^{\top,n-\frac{1}{2}}) F_{_F}^{-\top,n-\frac{1}{2}} \mathrm{Grad}\ \delta D_{_F}^{\top,n-\frac{1}{2}} F_{_F}^{-\top,n-\frac{1}{2}} \cdot N_{_F} \right\rangle_{\Gamma_{_{FSI}}}, \\
&+ \langle v_{_F},\ \delta L_{_U} \rangle_{\Gamma_{_{FSI}}} + \langle m_{_U},\ \delta U_{_F} - \delta U_{_S} \rangle_{\Gamma_{_{FSI}}} + \langle c_{_F},\ \delta L_{_D} \rangle_{\Gamma_{_{FSI}}} + \langle m_{_D},\ \delta D_{_F} - \delta D_{_S} \rangle_{\Gamma_{_{FSI}}}.
\end{aligned}
\tag{3.44}
$$

3.7 Description of the Jacobian Matrix $r'^{n,k}$

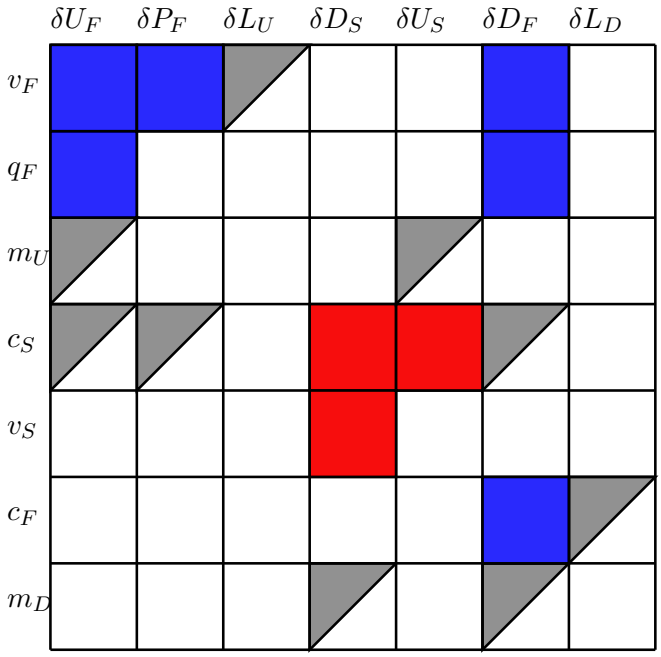

Figure 3.3: Sparsity pattern of the Jacobian matrix. Blue and red rectangles denote fluid and structure degrees of freedom respectively, including the interface. Grey triangles denote interface DOF's exclusively.

In Figure 3.3, a schematic view of the structure of the Jacobian $r'^{n,k}$ is presented in terms of trial and test functions. If we examine the variational forms of the Frechét derivative in Section 3.6, we can notice that each term in the form contains exactly one trial and one test function. During matrix assembly discretization is carried out according to these forms, and

the corresponding matrix entries are put into their respective blocks in the Jacobian matrix. Furthermore, the matrix has a *sparse* structure, meaning that the matrix contains many zero entries. Efficient storage of such matrices is provided by sparse linear algebra libraries, which do not explicitly store the zero entries.

Chapter 4

IMPLEMENTATION OF NEWTON'S METHOD FOR FSI

In this section the computer implementation of the FSI Newton's method algorithm, which was derived in Section 2, is described. In doing this we begin with a brief description of the most important technologies and external libraries which are used in the implementation. Next we give a presentation of FSINewton, the software that was created during this thesis project. Finally we end with a discussion of various factors which effect the run time and memory performance of FSINewton.

4.1 Technologies and External Libraries

A major part of the time spent on this thesis was used for the development and testing of FSINewton, the computer implementation of Newton's method for FSI. This solver was written in the Python programming language version 3.2.3, and relies upon the software package DOLFIN. Python and DOLFIN are described below, along with a few other software components and programming languages.

4.1.1 Python

The Python programming language was developed by Guido Van Rossum in the late 1980's, and has since then grown substantially in popularity, especially among scientific programmers. One of the main advantages of Python is that it enables faster development of software, due to it's compact syntax and the fact that it is a scripting language that does not require compilation. Two of the notable Python utility packages used in this thesis are PyPlot, which was used to create charts and graphs, and PyTest, which was used in the development of unit tests.

4.1.2 C++

C++ is a popular programming language developed by Bjarne Stroustrup, starting in 1979 at Bell Labs. It is a compiled language that is able to achieve good runtime speed due to low level optimizations made possible through strong typing and effective memory management. In the context of this thesis C++ code is generated by DOLFIN in order to efficiently assemble matrices and vectors.

4.1.3 DOLFIN

The DOLFIN software package is the main user interface of the core component of the FEniCS Project (`www.fenicsproject.org`), whose aim is the automated solution of mathematical models based on differential equations. DOLFIN provides a consistent problem solving environment that provides data structures and algorithms for finite element meshes, automated finite element assembly, and numerical linear algebra. DOLFIN integrates the following FEniCS components

- UFL (Unified Form Language), a domain-specific language embedded in Python for specifying finite element discretizations of differential equations in terms of finite element variational forms [Alnæs, 2012];
- FIAT (Finite element Automatic Tabulator), a Python module for generation of arbitrary order finite element basis functions on simplices [Kirby, 2004, 2012];
- FFC (FEniCS Form Compiler), a compiler for finite element variational forms taking UFL code as input and generating UFC output [Kirby and Logg, 2006, Logg et al., 2012b, Ølgaard and Wells, 2010];
- UFC (Unified Form-assembly Code), a C++ interface consisting of low-level functions for evaluating and assembling finite element variational forms [Alnæs et al., 2012, Alnaes et al., 2009];
- Instant, a Python module for inlining C and C++ code in Python;

PDE problems are specified in FEniCS using UFL, which closely mimics the mathematical notation for variational forms. The forms are then interpreted by FFC, and used to generate C++ code for the assembly of matrices and vectors. The discrete equations, which are based on the assembled matrices and vectors, can then be solved using one of the back-end linear algebra packages.

FEniCS brings together the speed of C++ with the elegant interfaces of Python. During the software development part of this thesis project, FEniCS's automatic finite element discretization capabilities proved to be

invaluable, as they substantially reduced the potential sources of error in the implementation.

4.1.4 PETSc

PETSc is the default linear algebra backend used by DOLFIN. It includes special data structures and functions for *sparse* linear algebra, meaning that the matrices that are involved have a large amount of zero entries. By taking the sparsity structure of a matrix in account, large amounts of memory and runtime can be saved. Many modern linear solve algorithms and preconditioners are implemented in this package, including a variety of LU and iterative solvers.

4.2 Description of FSINewton and CBC.Swing

FSINewton, the software package built in this thesis, is a collection of Python scripts that use C++ code generated by DOLFIN in order to do the necessary finite element calculations needed to solve an FSI problem with Newton's method. The architecture of FSINewton includes many object oriented features in the hope of making expansion and integration of the code easier for future programmers.

The code base of FSINewton is available as part of the open source package CBC.Solve, (`https://launchpad.net/cbc.solve/`), in the folder *cbc.solve/cbc/swing/fsinewton*. The files of FSINewton are included in the solver Swing, which is a framework for solving FSI problems that includes adaptive mesh refinement. In the Swing framework the user specified primal problem is solved either by fixed point iteration (see Selim [2012]), or by the Newton's method code of this thesis. In addition to the primal problem, Swing solves a so called dual problem, which is used for error control. The formulation of the dual problem includes the adjoint of the Fréchet derivative R'^n, which is derived in section (3.44). The same code is used for the specification of the variational forms of R'^n in FSINewton, and in the dual problem of CBC.Swing.

4.2.1 Top Level Components of FSINewton

The highest level of the FSINewton framework consists of two components, a user specified FSI problem of class NewtonFSI, and a solver of class FSINewtonSolver. The FSI problem is passed to the solver upon initialization of the solver, and the solver solves the problem via the method solve() using two loops; one for time stepping, and the other for Newton iteration. Figure 4.1 illustrates this framework.

As an alternative to the native problem class NewtonFSI, the subclass FSI included in CBC.Swing may also be used to define and solve FSI problems.

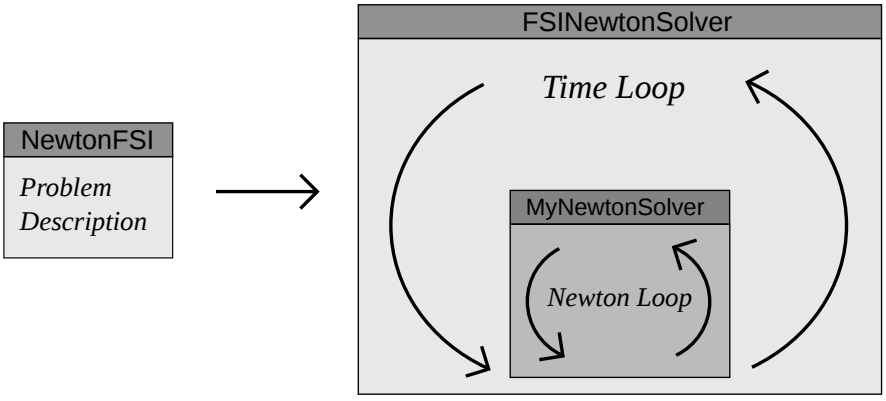

Figure 4.1: Top level description of the FSINewton framework. A user specified FSI problem of type NewtonFSI is given to the solver, which uses an outer time loop and an inner Newton loop in order to simulate the problem.

An example of this is given in the file *cbc.solve/demo/swing/analytic/newtonanalytic.py*.

4.2.2 Main Classes

The three top level classes of FSINewton are the problem class NewtonFSI, the FSI solver class FSINewtonSolver, and the Newton's method implementing class MyNewtonSolver.

Class NewtonFSI

The envisioned way to define a Newton's method FSI problem class is to derive it from the base class, NewtonFSI. The base class includes default methods that can be overwritten. These methods are used for the specification of material parameters, boundary conditions, initial conditions, forces, and various other parameters and tolerances that effect the performance and accuracy of the solver.

The definition of boundary conditions for the fluid problem should be done carefully, as the Newton's method fails to converge if any fluid velocity Dirichlet boundaries overlap with the interface Γ_{FSI}. Additionally, one should be careful when switching between the Newton's method and fixed-point solvers of CBC.Swing. The fluid equations are solved over the reference domain in the Newton formulation, and over the time dependant current domain in the fixed point formulation. This means that user specified fluid forces, boundaries and initial conditions need to take a domain mapping in account when switching bewteen Newton's method and fixed point solvers.

Class FSINewtonSolver

The class FSINewtonSolver contains the solution algorithm for problems of type NewtonFSI. The three most important methods of this class are __init__,solve, and time_step.

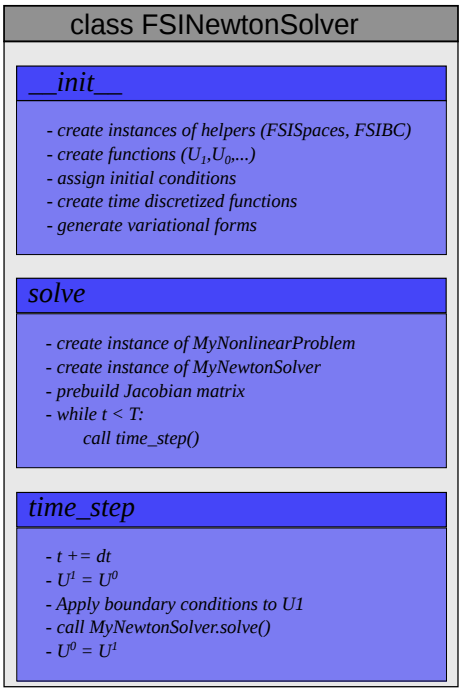

Figure 4.2: Description of class FSINewtonSolver.

method FSINewtonSolver.__init__ In Python, __init__ is the name given to the object constructor method of a class. For the class FSINewtonSolver, this method is responsible for the instantiation of helper classes FSISpaces and FSIBC, which are responsible for the handling of discrete function spaces and user specified Dirichlet boundary conditions, respectively.

The __init__ method also creates data structures corresponding to the system state, U^0 and U^1, at the previous and current time levels, and assigns the user specified initial conditions to U^0. Furthermore, the cG(1)time discretization scheme is applied here via the creation of symbols for time discretized functions. The symbols are then later inputed into the variational forms. Other time discretization schemes can be easily implemented here by changing the symbol definitions.

Finally, the creation of variational forms corresponding to the residual, R^n, and Frechét derivative R'^n, are also handled by the method __init__.

method FSINewtonSolver.solve The method FSINewtonSolver.solve triggers the solution of the problem given by an object of class NewtonFSI. In doing this the relevant data is saved as a MyNonLinearVariationalProblem, which is passed on to a MyNewtonSolver. At each step of the time loop a call to FSINewtonSolver.time_step is made.

method FSINewtonSolver.time_step At each time step the global time variable t is updated, and the initial guess $U^1 = U^0$ is set for the Newton's method algorithm. Boundary conditions are applied to U^1, which is necessary if the boundary conditions are time dependent. The current system state U^1 is then determined by a call to MyNewtonSolver.solve(). At the end of the time step the previous system state is updated, $U^0 = U^1$.

Class MyNewtonSolver

The class MyNewtonSolver is a general purpose Newton solver with a few extra features, namely plotting, Jacobian reuse and feedback in case of non convergence.

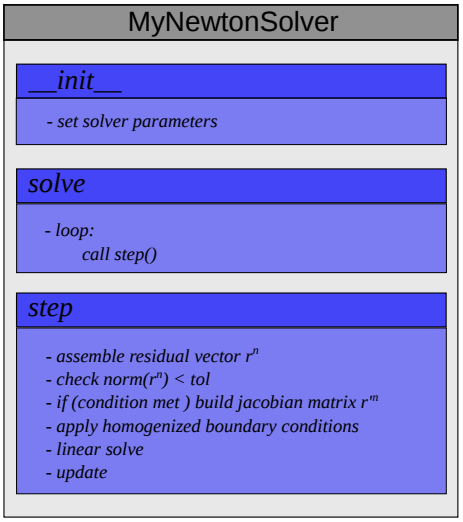

Figure 4.3: Description of class MyNewtonSolver.

method MyNewtonSolver.__init__ The method MyNewtonSolver.__init__ stores the information contained in the MyNonLinearVariationalProblem, and sets global Newton solver tolerance and Jacobian reuse parameters.

method MyNewtonSolver.solve The method MyNewtonSolver.solve, triggers the solution of the MyNonLinearVariationalProblem, in the FSI case giv-

ing the value of U^1. Newton iterations are carried out until the norm of the residual is less than the tolerance.

method MyNewtonSolver.newton_step In each Newton iteration the residual is first assembled, and then it's norm is checked for convergence. Next the Jacobian matrix is assembled. If Jacobian reuse is set the assembly is only carried out if a specified number of iterations have been done without reaching convergence. Next boundary conditions are applied to the residual vector and Jacobian matrix. These Dirichlet boundary conditions are homogenized as the inhomogeneous boundary conditions were already applied previously to U^1. Finally, the linear system $r'^n \delta\tilde{U}_k = -r^n$ is solved for the increment $\delta\tilde{U}_k = \tilde{U}^1_k - \tilde{U}^1_{k-1}$, and the degrees of freedom of the current system state are updated $\tilde{U}^1 += \delta\tilde{U}_k$.

4.2.3 FEniCS Implementation of the FSI Variational Forms

The forms of R'^n and R^n are written as symbols in Unified Form Language, and are converted into vectors and matrices at each time step via the DOLFIN function assemble(). Below is an excerpt of the code responsible for the creation of the residual form R_F.

Python code

```
def fluid_residual(Udot_F,U_F,U1_F,P_F,v_F,q_F,mu,
                   rho,D_F,N,dx_F,ds_DN,ds_F,F_F,Udot_M,
                       G_F=None):
    #Operators
    Dt_U = rho*J(D_F)*(Udot_F +
        dot(grad(U_F),dot(inv(F(D_F)),U_F - Udot_M)))
    Sigma_F = PiolaTransform(_Sigma_F(U_F, P_F, D_F, mu),
        D_F)
    #DT
    R_F  = inner(v_F, Dt_U)*dx_F
    #Div Sigma F
    R_F += inner(grad(v_F), Sigma_F)*dx_F
    #Incompressibility
    R_F += inner(q_F, div(J(D_F)*dot(inv(F(D_F)), U_F)))*dx_F
    #Use do nothing BC if specified
    if ds_DN is not None:
        info("Using Do nothing Fluid BC")
        R_F += -inner(v_F,
            J(D_F)*dot((mu*inv(F(D_F)).T*grad(U_F).T -
            P_F*I)*inv(F(D_F)).T, N))*ds_DN
    #Add boundary traction (sigma dot n) to fluid boundary
        if specified.
    if ds_F is not None and ds_F != []:
        info("Using Fluid boundary Traction Neumann BC")
        R_F += - inner(G_F, v_F)*ds_F
    #Right hand side Fluid (body force)
    if F_F is not None and F_F != []:
        info("Using Fluid body force")
        R_F += -inner(v_F,J(D_F)*F_F)*dx_F
    return R_F
```

In this function the time discretized symbols for the fluid velocity are passed via the method arguments Udot and U. Other residual forms are defined similarly and added together to yield R^n. The complete code can be found in the file *fsinewton/solver/residualforms.py*.

The forms corresponding to R'^n can also be generated via explicit definition in Unified Form Language, similarly to the forms of R^n. This is done in *fsinewton/solver/jacobianforms.py*. An alternative to this is the DOLFIN function derivative(), which can be used to derive the forms of R'^n via symbolic differentiation, as demonstrated by the code below.

Python code

```
j = derivative(r,self.U1)
```

This automatic differentiation was developed in Alnæs [2009].

The manually specified Frechét derivative provides more detailed control of the forms. In this thesis this was used in order to buffer linear parts of R'^n for increased performance, as described in section (6.2.3). The use of the automatically derived Frechét derivative however significantly reduces potential error in the code, as the manual derivation of R'^n is rather long and

detailed, (see Section 3.6). Automatically derivation also makes switching mathematical models much easier, as only the corresponding residuals need to be updated with the relevant variational forms. In order to check that the automatic and manual Frechét derivatives match, a automated test was written to check for any differences between the corresponding matrices. This test is described in the next section.

4.2.4 Integration Tests

During the development of FSINewton several Integration tests were developed to insure the quality of the code for future development. The three best tests are included in the CBC.solve package, in the folder */dev-vfsi/cbc.solve/test/swing/fsinewton*. These tests are implemented using the Python library py.test, which allows the convenient running of unit tests from the command line with the command

Bash code

```
1 cbc.solve/test/swing/fsinewton/unittests\$ py.test
```

The three unit tests are now briefly described.

test_jacobian.py

This test can be used to check the agreement of the Jacobian matrices resulting from the manual and automatic differentiation of R'^n, in addition to the one obtained when using the Jacobian buffering scheme. The entries of the manual and automatic Jacobians should agree up a tolerance of 10^{-14}. The buffered Jacobian only agrees up to a tolerance of 10^{-12} with the other two Jacobians. This could be due to a bug or possibly have something to do with the way the roundoff errors propagate during the addition of the buffered and unbuffered parts.

test_problemsetup.py

This file contains a sequence of unit tests to ensure that an FSI problem has been set up correctly. It checks among other things that the FSI interface has the expected length and that initial and boundary conditions match. Additional problems can be checked by the test by a few modifications to the unit test code.

test_analytic.py

Unlike the other two tests this one is not a part of the pytest framework. Instead running this file generate a convergence plot using the analytic test problem. This is the most important check that insures that FSINewton still works. If the convergence plots show some sort of anomalies then further

investigation is possible by experimenting with the prescription of the exact solution to the fluid or structure variables.

4.3 Factors Effecting the Memory Use and Runtime of FSINewton

The two most important computational resources for a computer simulation are memory and runtime, as these limit the precision of the simulation and the complexity and size of what can be simulated. In the case of FSINewton, the major factors influencing memory and run time are hardware, system size and composition, choice of linear solver, the state of the FEniCS Form Compiler, and the lack of function space restriction.

Hardware

The computer hardware running the FSINewton code can be expected to have a large influence on performance. Roughly speaking, processor speed greatly influences the program run time, and memory limits the size of the matrices that can be stored, and thereby the size of the linear systems that can be solved during each Newton iteration. All of the simulations and data gathering in this thesis were done using a Lenovo W520, whose specifications are given in Table 4.1.

Table 4.1: Specifications of the laptop (Lenovo W520) used to gather run time data

Component	**Spec**
chipset/cpu	Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8
Grahics card	NVIDIA® QUADRO® 1000M 2GB VRAM with 96 CUDA CORES
memory	15.6 GB
Operating system	Ubuntu 12.04 LTS
System architecture	64 bit

4.3.1 System size and system composition

The size of the linear system in the Newton's method formulation of the FSI problem is the fundamental factor influencing the computational cost of the solution. As the dimension of the linear system increases, so does the runtime needed for the assembly and solution of the system. Additionally, a larger system requires more memory for matrix storage.

Mesh fineness and geometry have a direct influence on the size of the discrete linear system. Finer meshes containing more elements lead to bigger systems than coarse meshes with less elements. Mesh connectivity also plays

a role in the case of continuous Galerkin formulations, as elements sharing facets will also share degrees of freedom. However, the cost of every mesh entity is not the same. For example fluid cells are typically more expensive than structure cells due to the presence of the domain mapping.

It is also worth noting that FSI interface facets are more expensive than non interface facets. This is because the interface facets appear in the variational forms related to every single test and trial function of the functional Newton's method, equation 3.25, thereby generating the maximum possible global degrees of freedom in the discrete system matrix. In comparison, a fluid or structure facet will appear only in the corresponding fluid or structure variational forms; as part of a cell integral.

Table 4.2 summarizes the cost of a single mesh entity with the standard function space configuration. This information can be used to get a feel for

Table 4.2: DOF's generated by a single mesh entity in the standard function space configuration of FSINewton. Here d is the spatial dimension and p the polynomial degree of the structure velocity, structure displacement, and fluid domain displacement fields.

d	**p**	Fluid Cell DOF's	Structure Cell DOF's	Interface Facet DOF's
2	**1**	21	12	28
2	**2**	27	24	34
3	**1**	49	24	60
3	**2**	70	66	87

the relative costs of cells and facets. It is is however, not well suited to estimating the global function space dimension since many degrees of freedom are shared by neighbouring entities. More precise information regarding system composition is delivered by the automatically given by the FSINewton code before beginning a simulation.

4.3.2 Choice of linear solver

The solution of the linear system of equations resulting from the discrete Newton's method is one of the core operations of the solution algorithm used in FSINewton. Typically a large number of linear solves are carried out as the linear solve routine is placed inside both the Newton and time loops. The current implementation of FSINewton uses LU factorization to solve linear systems. The LU factorization involves the use of Gaussian elimination to factorize a matrix into upper and lower triangular matrices, which are then used to solve the linear system using backwards and forwards substitution.

One disadvantage of the LU solver is that the memory footprint of the factorized matrix might be larger than that of the original matrix, as zero

entries can be filled in. A simple example of LU fill-in is shown below.

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \xrightarrow{LU\ \ factorization} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix} \tag{4.1}$$

The factorized matrix on the right is $(L + U - I)$, which can be stored slightly more efficiently than L and U seperately. The factorized matrix is still larger than the original matrix on the left however, as the factorized matrix contains an additional zero, thereby using more memory in sparse storage.

The alternative to LU factorization are the so called *Krylov* methods, which use an iterative procedure to approximate the solution of the system. These methods are typically faster than direct methods for larger system sizes, and are also more memory efficient as they avoid the fill in problem. At the moment none of the standard Krylov method - preconditioner combinations available with the software package PETSc converge, meaning that LU is unfortunately the only viable linear solver choice for FSINewton at the moment.

In Heil [2004] a Krylov method is successfully implemented using a custom made block-preconditioning scheme. The FSI model used in this article is very similar to the one used here, and it is highly likely that the same scheme, possibly with slight modifications, could be applied to FSINewton.

4.3.3 FEniCS Form Compiler

The FEniCS Form Compiler (FFC) is responsible for taking user specified variational forms and then generating C++ code for the efficient assembly of the required matrices and vectors. The assembly of the FSI Jacobian matrix and residual vector is done with so called quadrature representation, meaning that the matrix and vector entries are calculated using numerical integration over a set of quadrature points. An alternative to this is the so called tensor representation, which calculates local tensor contributions using the contraction of a constant reference tensor with a variable geometry tensor. At the moment only quadrature representation is viable for the FSINewton code.

FFC already contains a wide variety of optimizations, and others are in constant development. Any changes in the FFC technology can be expected to influence the matrix and vector assembly times of the FSINewton code. For a good overview of FFC see Alnæs [2012].

4.3.4 Function Space Restriction

Currently there is no way to restrict function spaces to a certain part of the mesh using DOLFIN. The consequence of this is that functions in FSINewton are defined over an entire mesh, and that the Jacobian matrix includes

a great deal of ‘dead‘ degrees of freedom whose corresponding rows and columns are equal to 0. In order to get a nonsingular system, a work around is used that replaces the diagonal entries of zero rows with ones. The full effects of this on the performance of FSINewton are unknown. Assembly time is probably unaffected, as the variational forms that are assembled are only defined over their proper domain. Linear solver time however is probably slowed down a bit due to the addition of many unnecessary dimensions in the linear system. The good news is that function space restriction is currently under development, and it should soon be possible to fix this defect.

Chapter 5

TEST PROBLEMS

This section contains the description and results of the simulation of three test problems, each of which represents an increase in computational difficulty over the previous problem. The first test problem features a known analytical solution, which was specifically manufactured to check the accuracy of the implementation. The second problem is a channel with a flap, representative of 'easy' FSI problems which involve a rather dense structure. The third is a 2-D slice of an idealized blood vessel which is a 'challenging' FSI problem due to the similar densities of the structure and fluid, and longer FSI interface. These problems are presented individually in the following sections, following a brief discussion of what makes an FSI problem challenging to simulate.

5.1 Computational Considerations For FSI With The ALE Method

The computational difficulty of an FSI problem using the ALE method is determined by several factors:

1. Magnitude of displacement along the FSI interface
Larger displacements of the FSI interface are harder to simulate due to the greater possibility of mesh elements degenerating under the fluid domain mapping.

2. Size and density of the FSI interface relative to the computational domain
FSI interfaces which are larger and more dense have a greater influence on the fluid domain geometry and are therefore harder to simulate. Dense interfaces especially challenge the fluid mapping as they leave less room for smoothing.

3. Relative density of the fluid and structure
The sensitivity of the fluid and structure variables increases as the two densities approach one another. This challenges the ability of numerical methods to obtain convergence.

4. Reynold's number of the fluid
The Reynold's number of an incompressible flow measures the ratio of inertial to viscous forces in the fluid. It is defined as $Re = \frac{\rho u L}{\mu}$. Where ρ is the density of the fluid, u the mean velocity, L a characteristic length, and μ the dynamic viscosity of the fluid. Higher Reynold's number flows are harder to simulate due to possible turbulences in the flow. In this thesis only low Reynold's number flows are considered, i.e. so called *laminar* flows. For more information about the Reynold's number see White [2005].

5.2 The Analytical Problem

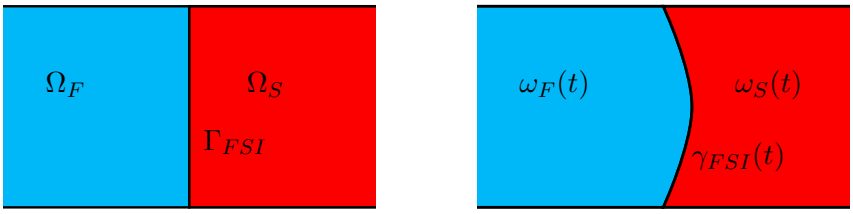

Figure 5.1: Geometry of the manufactured solution described in Section 5.2.

The analytical problem is posed on a rectangular region $\Omega = \{x, y|\ 0 < x < 2, 0 < y < 1\}$ divided into two squares corresponding to the fluid domain $\Omega_F = \{(x, y)|\ 0 < x < 1, 0 < y < 1\}$ on the left and the structure domain $\Omega_S = \{(x, y)|\ 1 < x < 2, 0 < y < 1\}$ on the right. Displacement and velocity fields were chosen to hold across both domains, along with a fluid pressure, as detailed in Table 5.1. Using this information, the other variables of interest could be derived using the symbolic differentiation tools in the Python package SymPy. The resulting boundary tractions for the fluid and structure do not match on Γ_{FSI}, which is compensated for by adding an additional traction term,

$$\langle c_S,\ G_{FSI} \rangle_{\Gamma_{FSI}} \tag{5.1}$$

to the Unified Form Language (UFL) specification of the total FSI residual (3.18). The analytical problem also depends on a scaling parameter C which is set to $C = 2$ in our numerical experiments.

The term G_F, which is the fluid stress on the left side of the fluid domain, is calculated so that a Neumann boundary condition can be used in the fluid

Table 5.1: Parameter and coefficient values for the Analytic FSI problem.

Material parameters	$\rho_F = 1, \rho_S = 100, \mu_F = 1, \mu_S = 1, \mu_F = 1, \lambda_S = 2, \lambda_F = 2$
Function values	$U_{_F} = y(1-y)\sin t$ $P_{_F} = 2C\sin t(1 - x - Cxy(1-y)(1-\cos t))$ $D_{_S} = Cy(1-y)(1-\cos t)$ $U_{_S} = Cy(1-y)\sin t$ $D_{_F} = Cxy(1-y)(1-\cos t)$
Body forces	$B^x_{_F} = J_{_F} Cy(1-y)\cos t$ $B^y_{_F} = 0$ $B^x_{_S} = -2C^3(6\cos t - 6)(2y\cos t - 2y - \cos t + 1)^2$ $\quad + C(100y(1-y)\cos t - (2\cos t - 2)$ $B^y_{_S} = 8C^2(2y\sin^2 t + 4y\cos t - 4y - \sin^2 t - 2\cos t + 2)$ $B^x_{_{FD}} = Cx(-y^2\sin(t) + y\sin(t) - 2\cos(t) + 2)$ $B^y_{_{FD}} = 3C(-2y\cos t + 2y + \cos t - 1)$
Boundary forces	$G^x_F = \sin t(2C(-Cxy(1-y)(1-\cos t) - x + 1)$ $G^y_F = -C(1-2y)\sin t)$ $G^x_{FSI} = C(6Cxy^2\cos t - 6Cxy^2 - 6Cxy\cos t$ $\quad + 6Cxy + Cx\cos t - Cx + 2x - 2)\sin t$ $G^y_{FSI} = C(4C^2x^2y^3\sin^2 t + 8C^2x^2y^3\cos t - 8C^2x^2y^3$ $\quad - 6C^2x^2y^2\sin^2 t - 12C^2x^2y^2\cos t + 12C^2x^2y^2$ $\quad + 2C^2x^2y\sin^2 t + 4C^2x^2y\cos t - 4C^2x^2y$ $\quad - 4Cx^2y\cos t + 4Cx^2y + 2Cx^2\cos t$ $\quad - 2Cx^2 + 4Cxy\cos t - 4Cxy$ $\quad - 2Cx\cos t + 2Cx - 2y + 1)\sin t$

problem, which makes the pressure unique. Also, G^x_{FSI}, G^y_{FSI} are the x and y components of the difference between the structure and fluid traction vectors on $\Gamma_{_{FSI}}$.

5.2.1 Convergence of the kinematic variables

Convergence testing was carried out with the FEniCS implementation of Newton's method for the FSI analytical problem, on the time interval $[0, 0.1]$ with constant time step $k^n = 0.02$ and a coarse initial mesh consisting of (10×5) squares, each divided into two triangles along the diagonal from bottom left to top right. In each time step, Newton's method was carried out until the norm of the residual vector $\|b(U^n_k)\|$ was less than the tolerance TOL $= 10^{-13}$. At each subsequent refinement level, the time step was halved, and the mesh refined uniformly. The results are plotted in Figure 5.2, which uses first order polynomial approximation of the variables $U_{_F}$,$P_{_F}$,$U_{_S}$

and Figure 5.3, which uses second order polynomial approximation of the variables U_F,P_F,U_S. These two choices correspond to $p = 1$ and $p = 2$ in Table 3.1.

The mixed formulation with $p = 1$ exhibits good convergence behavior with a convergence order of around 2.5 for U_F, 2 for D_S and D_F, and order 1 for P_F. For the second mixed formulation, $p = 2$, the errors are smaller but the convergence rates are also seemingly lower. Here 1st order convergence was observed for U_F,P_F,U_S and 2nd order was observed for D_F. The paradoxical lower order convergence rates with the higher order discretizations is most likely due to the error in the case $p = 2$ being dominated by the time discretization error.

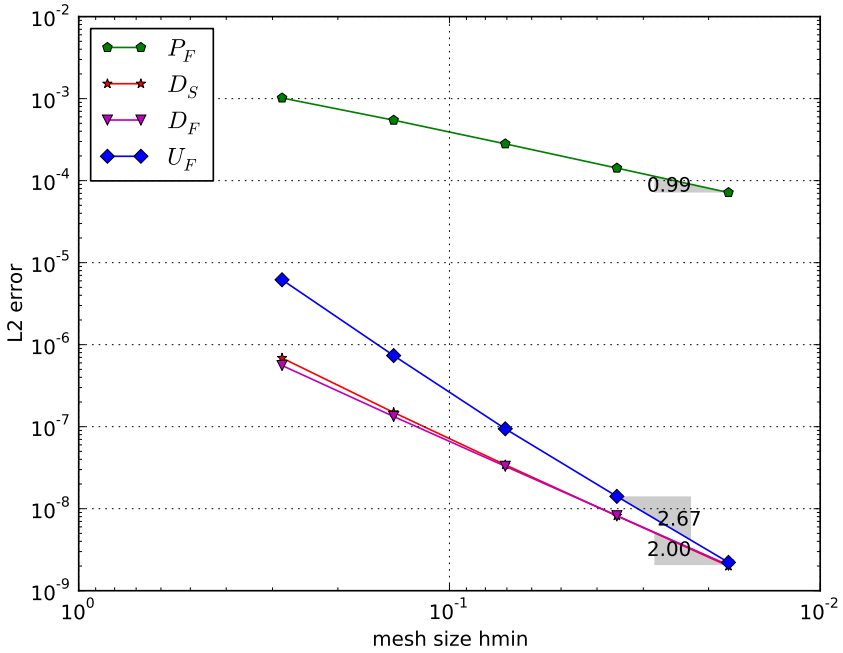

Figure 5.2: Convergence for the analytic test problem using first *order polynomial approximation for the structure and fluid domain variables, D_S, U_S, D_F. The order of polynomial approximation is two for the fluid velocity U_F, and one for the fluid pressure P_F.*

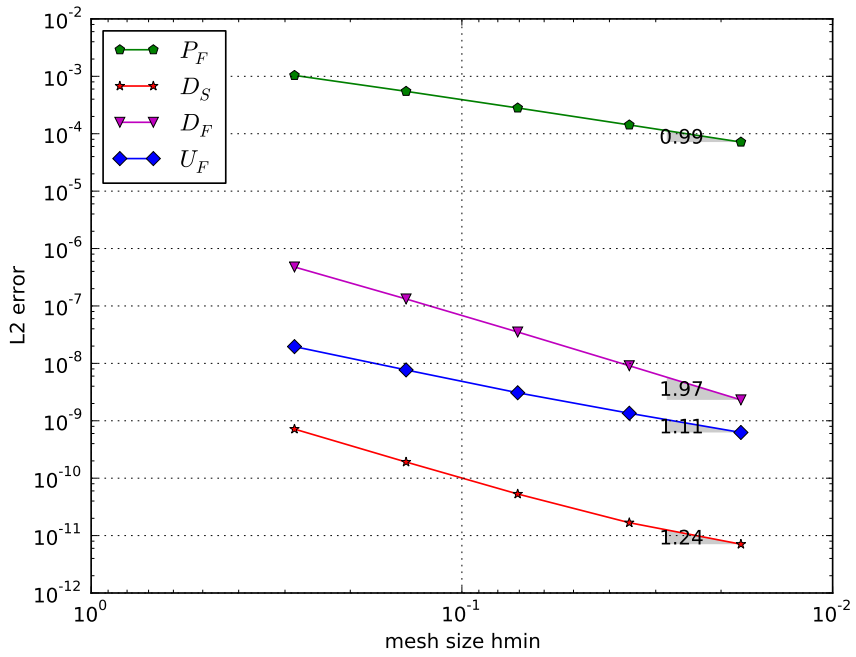

Figure 5.3: Convergence for the analytic test problem using second *order polynomial approximation for the structure and fluid domain variables, D_S, U_S, D_F. The order of polynomial approximation is two for the fluid velocity U_F, and one for the fluid pressure P_F.*

5.2.2 Convergence of the Lagrange multiplier conditions

With increased mesh and time step refinement it is to be expected that the interface conditions, $U_F - U_S = 0 = D_F - D_S$, enforced by the Lagrange multipliers will be met with increased precision. When examining this we first consider the FSI interface L^2 norm, $\|\cdot\|_{\Gamma_{FSI}}$. The convergence data in this norm is presented in Figure 5.4. The parameter p indicates the degree of the local polynomial approximation of (D_S, U_S, D_F).

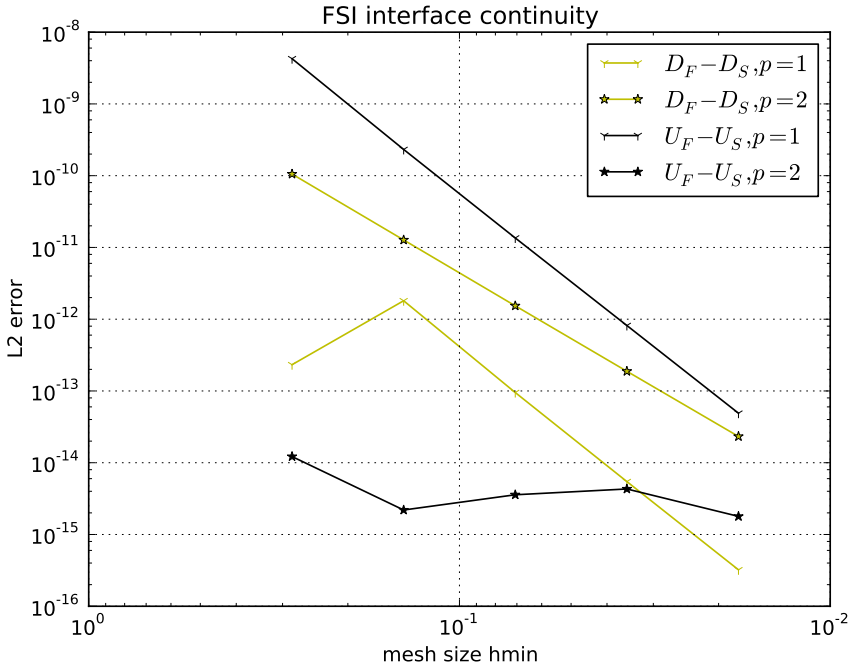

Figure 5.4: FSI Interface Continuity L2 error. p denotes the polynomial order of the structure displacement, structure velocity, and fluid domain displacement.

The L^2 error plot shows an overall trend of decreased overall error with increased mesh refinement. The first exception here is the increase in error of $D_F - D_S, p = 1$ in the second refinement, whose cause is unknown. The second exception is the error in $U_F - U_S, p = 2$, whose error is so close to the double floating point precision of 10^{-16} that it is negligible.

From the L^2 error information we can observe that the convergence rates for $D_F - D_S, p = 1$ and $U_F - U_S, p = 1$ are about order 4. For $D_F - D_S, p = 2$ we observe 3 order convergence, and of course no convergence for the already small $U_F - U_S$.

We next consider the discrete relative error of the Lagrange multiplier

conditions, which is given by

$$e_{\text{rel}}(f_1, f_2) = \max_{x \in \tilde{\Gamma}_{FSI}} \left\{ \frac{|f_1(x) - f_2(x)|}{\max\{|f_1(x)|, |f_2(x)|\}} \right\}. \tag{5.2}$$

Here $\tilde{\Gamma}_{FSI}$ is the set of mesh nodes along the FSI interface. Figure 5.5 shows the plots containing the new error of the Lagrange multiplier conditions.

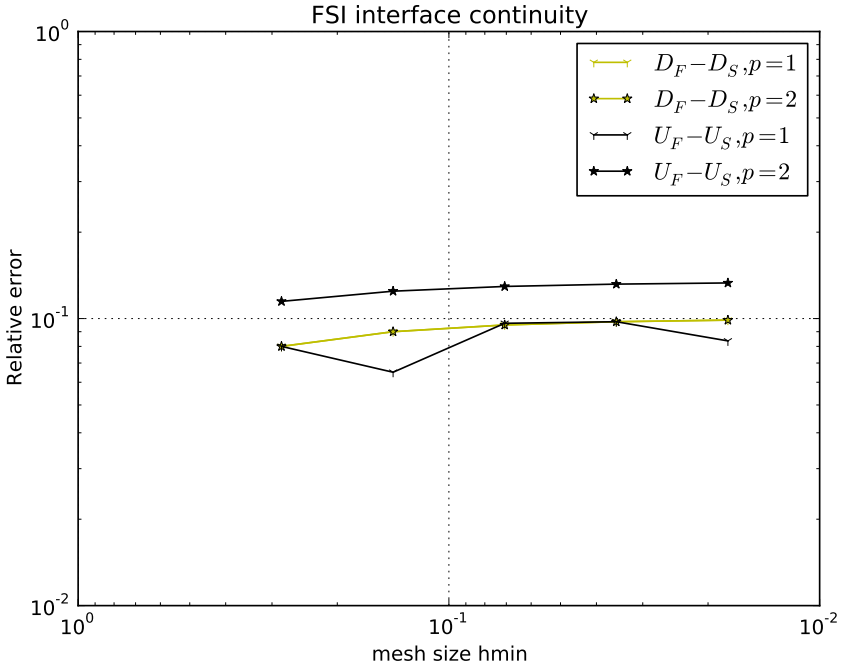

Figure 5.5: FSI Interface Continuity relative error. Here p denotes the polynomial order of the structure displacement, structure velocity, and fluid domain displacement.

The max relative errors tells us something about the worst that the error can get. In the cases $p = 1, p = 2$ it appears that the continuity error is never much worse than about 10% relative to the displacement and velocity of the interface. This might not be good enough for some applications. Also the discrete relative error doesn't decrease substantially with mesh refinement. Experimenting with changing the Lagrange multiplier function spaces to cG(1) didn't improve this defect.

5.3 Channel With a Flap: An Easy FSI Problem

The channel with a flap problem is comprised of a fluid flowing through a channel which is partially blocked by a hyperelastic flap. This problem,

although a little more complex than the analytical problem, is still relatively easy to simulate due to the high structure density, small structure displacements, and relatively small FSI interface. This problem was taken from Selim [2011], and it's details are given in Table 5.2.

Table 5.2: Specification of the channel with flap problem.

Solution Domains	$\Omega = \{x, y \mid 0 \leq x \leq 4, 0 \leq y \leq 1\}$ $\Omega_{_S} = \{x, y \in \Omega \mid 1.4 \leq x \leq 1.8, y \leq 0.6\}$ $\Omega_{_F} = \Omega \setminus \Omega_{_S}$
Fluid Parameters	$\rho_F = 1$, $\mu_F = 0.001$
Structure Parameters	$\rho_S = 100$, $E_S = 100$, $\nu_S = 0.3$
Fluid Domain Parameters	$\mu_D = 1$, $\lambda_D = 1$
Initial Conditions	$U_{_F} = D_{_S} = U_{_S} = D_{_F} = (0, 0)$ $P_{_F} = 10(1 - x/4)$
Boundary Conditions	$\Gamma_{DN} = \partial\Omega_{_F} \setminus \Gamma_{_{FSI}}$ $P_{_F} = \begin{cases} 0 & \text{if} \quad x = 0 \\ 10 & \text{if} \quad x = 4 \end{cases} \Gamma_{_{FSI}}$ $D_{_S} = (0, 0)$ for $x, y \in \partial\Omega_{_S} \setminus \Gamma_{_{FSI}}$ $D_{_F} = (0, 0)$ for $x, y \in \partial\Omega_{_F} \setminus \Gamma_{_{FSI}}$ $D_{_S} = (0, 0)$ for $x, y \in \partial\Omega_{_S} \bigcap \partial\Omega$
Time	$0 < t < 0.1$, $k^n = 0.0025$
Mesh	5(width) x 20(length) grid with squares with side length 0.2, each square divided by a right diagonal into two triangles.

The structure material parameters of the problem are given in terms of *Young's modulus* E, which represents the stiffness of an elastic material, and also *Poisson's ratio* ν, which models the so called Poisson effect in the material. The Poisson effect is the tendency of a material to expand in the directions perpendicular to a compression. E and ν can be related to the Lamé parameters μ, λ by the relation

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{5.3}$$

The state of the problem at the end time is shown in Figure 5.6; at this point the flow of the fluid around the flap has developed, and a very slight compression of the flap has occured. At each time step the tolerance 10^{-6} was used for the Newton solver.

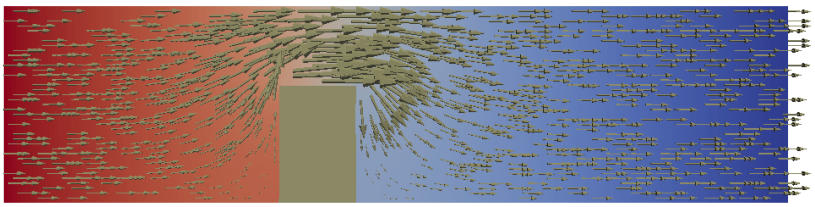

Figure 5.6: State of channel with a flap at end time $t = 0.1$. The arrows represent the fluid velocity, and the colour the fluid pressure. Areas coloured in red have higher pressure than those coloured in blue.

5.4 2D Blood Vessel: A Harder FSI problem

Biomedical FSI problems are an area of application where it is advantageous to use a full Newton's method. Biomedical solid materials often have densities which are close to those of fluids, which leads to FSI problems where the fluid and structure variables are more sensitive to one another. In this case, the full Newton's method may converge in instances where a quasi-Newton method or a fixed point iterations scheme may not. This was demonstrated in Fernández and Moubachir [2005].

Table 5.3: Specification of the blood vessel test problem.

Solution Domains	$\Omega = \{x, y \mid 0 < x < 6, 0 < y < 1\}$ $\Omega_{_F} = \{x, y \in \Omega \mid 0.1 < y < 0.9\}$ $\Omega_{_S} = \{x, y \in \Omega \mid y < 0.1 \text{ or } y \geq 0.9\}$
Fluid Parameters	$\rho_F = 1$, $\mu_F = 0.002$
Structure Parameters	$\rho_S = 4$, $\mu_S = 5$, $\lambda_S = 2$
Fluid Domain Parameters	$\mu_D = 100$, $\lambda_D = 100$
Initial Conditions	$U_{_F} = D_{_S} = U_{_S} = D_{_F} = (0, 0)$ $P_{_F} = 0$
Boundary Conditions	$\Gamma_{DN} = \{x, y \mid x, y \in \partial\Omega_{_F} \setminus \Gamma_{_{FSI}}\}$ $P_{_F} = P_F^*$ if $x = 0\Gamma_{_{FSI}}$ $D_{_S} = (0, 0)$ for $x, y \in \partial\Omega_{_S} \setminus \Gamma_{_{FSI}}$ $D_{_F} = (0, 0)$ for $x, y \in \partial\Omega_{_F} \setminus \Gamma_{_{FSI}}$
Time	$0 < t < 70,\ k^n = 0.5$
Mesh	10(width) x 60(length) grid with squares with side length 0.1, each square divided by a right diagonal into two triangles.

We here present one such problem, the simulation of blood flow in an idealized vessel, depicted in Figure 5.7 and described in Table 5.3. The flow is driven by a pressure wave p_F^* given at top and bottom boundary of the

inside of the vessel :

$$p_F^* = \begin{cases} 0 & \text{if} \quad x < 0.25t - 0.2 \\ C\cos(\frac{0.2\pi}{2}(x - 0.25t)) & \text{if} \quad |x - 0.25| \leq 0.2 \\ 0 & \text{if} \quad x > 0.25t + 0.2. \end{cases} \tag{5.4}$$

The resulting pressure wave causes the vessel walls to expand in a wave moving from left to right. Once the pressure wave passes the vessel begins to contract again. The numerical solution was obtained using a Newton solver tolerance of 10^{-6}

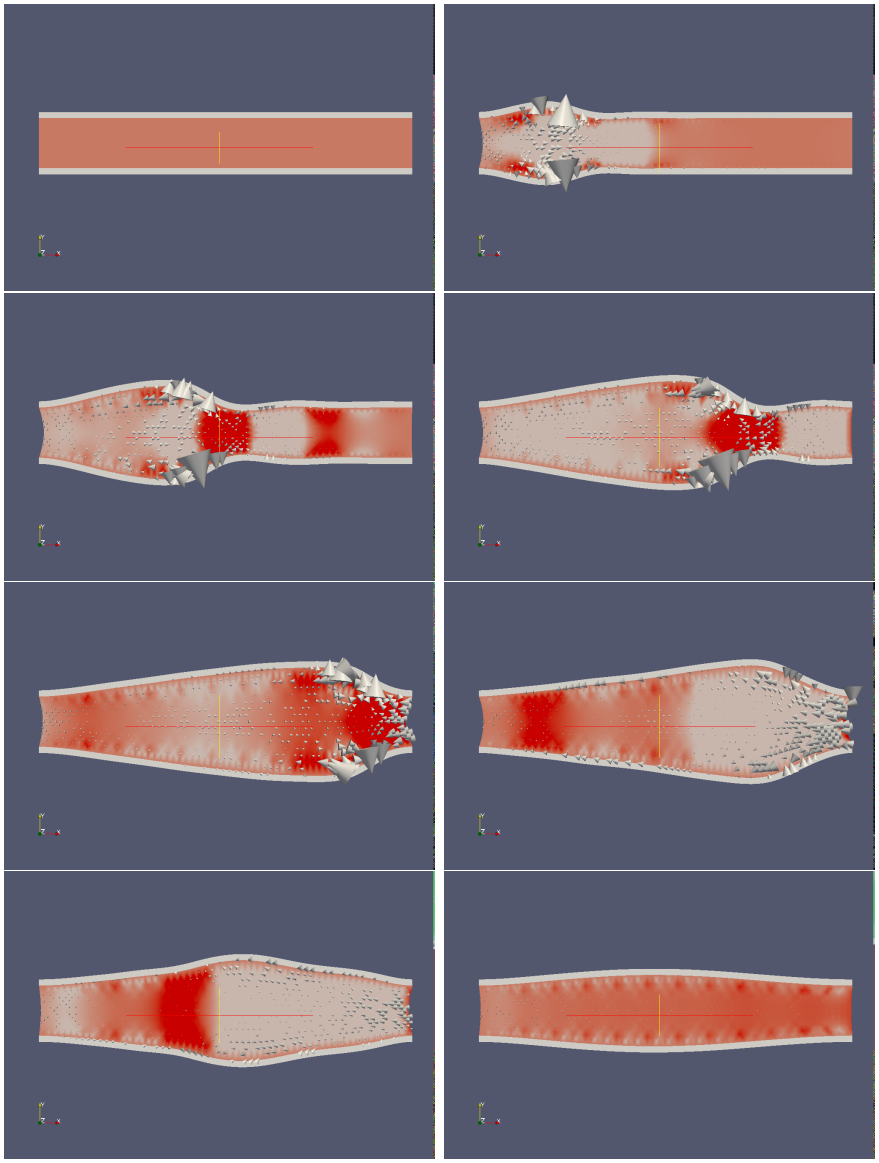

Figure 5.7: Blood vessel simulation at $t = 0, 5, 10, 15, 20, 25, 30, 70$.

Chapter 6

RUN TIME PROFILING AND OPTIMIZATION

FSI simulations have traditionally placed large demands on computer resources, and it for this reason that most computational FSI work has only been possible in the last decade. Performance was certainly an issue during the running of the three test problems, with the runtime often going into several hours, even for the moderate system sizes used. These long runtimes might be acceptable if one knows exactly what is to be simulated. However, more often than not one is interested in running many slightly varying simulations. As an example an engineer might search for an airplane wing design that optimizes lift and drag forces, leading to the carrying out of many simulations with slight changes to geometry, material composition and boundary conditions.

If a computer code is to be of practical use, it must be executable in a reasonable amount of time. It is for this reason that a quest for optimization was carried out with the FSINewton code. Various runtime data were gathered using the four test problems, and four optimizations were found, each suitable for use under a different set of circumstances. Memory use was only marginally considered as it wasn't yet an issue, but the simulation of even larger three dimensional systems sizes will certainly require some memory optimizations as well.

6.1 Profiling of the Standard Newton's Method

For the standard Newton's method, the number of Newton iterations is small when solving the three test problems described in Section 5. Only about 2-3 are needed to resolve each time step. It was observed that the value of the residual typically begins in the 1.0-2.0 range, and that the expected quadratic convergence predicted by Kantorovitche's Theorem is obtained [Kan, 1948].

Runtime data was gathered for the four test problems using the stan-

dard Newton's method with the Newton iteration tolerance set to 1.0^{-14} for the analytical problem and 1.0^{-6} for the other two problems. This data is presented in Table 6.1 with a breakdown of the runtime into the three dominant contributions; Jacobian assembly, linear solve and residual assembly. The overhead from the other parts of the FSINewton framework are negligible for the Channel with flap and Bloodvessel problem, but are significant for the small Analytical problem. As the Channel with flap problem and Bloodvessel problem represent more realistic system sizes than the analytical problem, the overhead has been ignored in the run time analysis.

Table 6.1: Test problem runtimes with standard Newton's method, and no optimizations. Mesh sizes are given in number of vertices

Problem	**Method**	**Calls**	**Run-time(s)**	**Run-time(%)**
Analytic problem	Jacobian assembly:	28	83.9286 s	90%
mesh vertices = 231	Linear solve:	28	1.8587 s	2%
time steps = 10	Residual assembly:	38	0.9152 s	1%
Channel with flap	Jacobian assembly:	80	13779.647s	95 %
mesh vertices = 6601	Linear solve:	80	552.659s	4%
time steps = 40	Residual assembly:	120	86.885s	1 %
2D Blood vessel	Jacobian assembly:	343	2978.8585 s	81%
mesh vertices = 1271	Linear solve:	343	2540.0204 s	18%
time steps = 140	Residual assembly:	483	64.1413 s	1%

The data shows a clear dominance of Jacobian assembly in the runtime of the test problems using the standard Newton's method. In all three problems, the Jacobian assembly time takes up at least 80% of the total runtime.

Linear solve is the next most expensive operation for the two larger test problems, Channel with flap and 2D Blood vessel. The dominance of Jacobian assembly time in the analytic problem is most likely explained by the small system size, which allows for relatively fast linear solves (see Figure 6.2 in the upcoming section).

High relative Jacobian assembly times can also be expected for systems which have many fluid degrees of freedom, such as the Channel with the flap. In Table 6.2, the system compositions are shown. One can see that the Channel with the Flap problem contains relatively more fluid degrees of freedom than the 2D Blood vessel problem. As the the more complicated forms of the fluid equation require higher order quadrature than that of the structure equation, one expects the Channel with the Flap problem to spend

a greater percentage of runtime in assembly relative to the 2D Blood Vessel. Table 6.1 shows that this is indeed the case when solving the problems using the unoptimized Newton's method.

Table 6.2: System compositions of the four test problems, measured in degrees of freedom. Fluid and Structure DOFs are here those that lie exclusively in the fluid and structure domains. The interface DOFs are measured separately.

Problem	Fluid	Structure	Interface	total DOFs
Analytic problem	60%	23%	16%	514
Channel with flap	97%	2%	1%	69705
2D Blood vessel	79%	7%	14 %	12981

6.1.1 Computational effort and mesh size

When considering the performance of a finite element algorithm, it is interesting to see how the runtime scales with the mesh size. In order to test this with FSINewton the analytic test problem was solved for a range of mesh sizes, using a constant time step length. The results are plotted in figure 6.2.

Reading from the graph, it appears that the time spent in Jacobian assembly and residual assembly grow at about a linear rate for the range of mesh sizes used. The linear solve time seems to be growing a little faster than linearly, and if the data were extrapolated one would see the linear solve time eventually dominate. It was noticed that more Newton iterations were required to solve the same time step with a finer mesh.

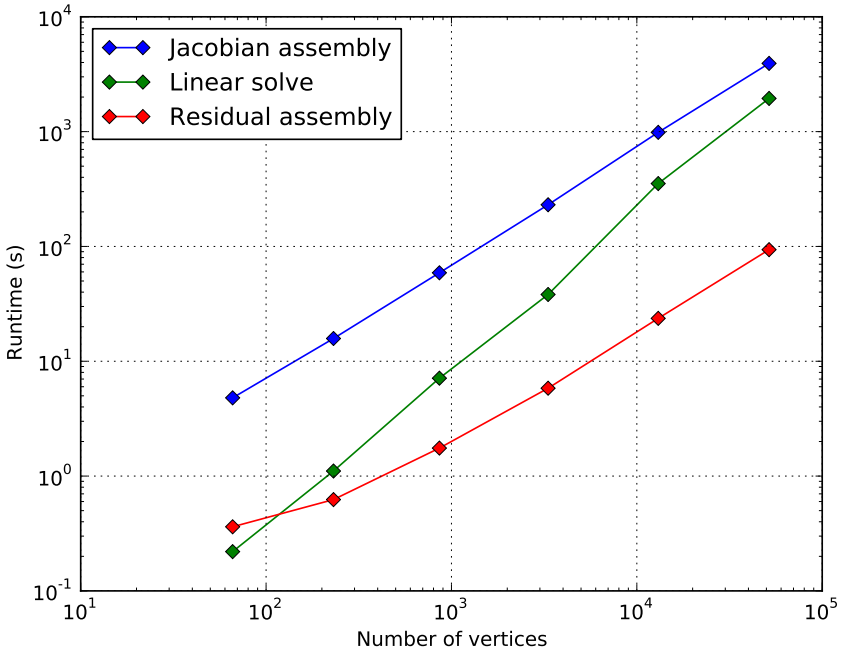

Figure 6.1: Timings vs. mesh size for the analytic problem with the number and size of time steps kept constant.

6.2 Optimization of FSINewton

After having discussed some considerations and examined the performance of the standard Newton's method we are now ready to make some optimizations to the algorithm. The best optimization is presented first, the reuse of the Jacobian. After that the technique of Jacobian simplification is presented, then a buffering scheme, and finally reduced order integration. All numerical experiments are done with the same Newton iteration tolerances as those used for the standard Newton's method.

6.2.1 Jacobian reuse

In Section 6.1 it was noted that Jacobian assembly is the most time consuming part of the FSINewton code when running the test problems. In many typical problems the system does not change dramatically over the course of a single time step, meaning that the corresponding changes to the Jacobian are also small. This means that the same Jacobian can be used to approximate the Jacobians of succeeding Newton iterations. This saves not only on assembly time, but also on linear solve time since the LU factorization can also be reused, making subsequent linear solves a simple matter of forward and backward substitution.

When running FSINewton with Jacobian reuse, Jacobian reassembly is only carried out if the number of Newton iterations in a time step reaches a user defined threshold. It was noticed however that a stale Jacobian could sometime cause a blow up in the Newton's method in cases where a fresh Jacobian would converge. In order to prevent this an extra check has been included in the code that triggers a Jacobian reassembly in the case that the size of the residual increases over a Newton iteration.

The results of running the test problems with Jacobian reuse are summarized in Table 6.3. The results show a dramatic reduction in runtime due to fewer Jacobian assemblies and cheaper linear solves. In all three problems the Jacobian assembly time has been reduced by at least 90%. As a trade off more residual assemblies are required. This added somewhere between 40%-192% to the residual assembly times. However the total run time is still reduced by at least %90 in the cases that were tested. It is worth noting that the Channel with flap and analytic problem were solved using a single Jacobian, whereas the more complicated 2D Blood vessel required some Jacobian reassemblies.

In Figure 6.2 the number of Newton iterations for a given time step are plotted for the 2D blood vessel problem. The run time data shows that 25 Jacobian reassemblies were necessary, while the graph only shows that two time steps have an iteration count above the threshold of 30, meaning that the remaining 23 reassemblies were due to blow ups in the residual. The relatively similar fluid and structure densities are probably responsible for

Table 6.3: Test problem runtimes with Jacobian reuse after 30 iterations.

Problem	**Method**	**Calls**	**% runtime**
Analytic problem	Jacobian assembly:	1 (-27)	-95 %
mesh vertices = 231	Linear solve:	51 (+23)	-96%
time steps = 10	Residual assembly:	61 (+23)	+54%
	Total Runtime:		**-93 %**
Channel with flap	Jacobian assembly:	1 (-79)	-99%
mesh vertices = 6601	Linear solve:	133 (+53)	-98%
time steps = 40	Residual assembly:	173 (+53)	+44%
	Total Runtime:		**-98%**
2D Blood vessel	Jacobian assembly:	25 (-308)	-93 %
mesh vertices = 1271	Linear solve:	1287 (+944)	-92 %
time steps = 140	Residual assembly:	1427 (+944)	+192%
	Total Runtime:		**-91 %**

the increased challenge posed to the convergence of the Newton's method.

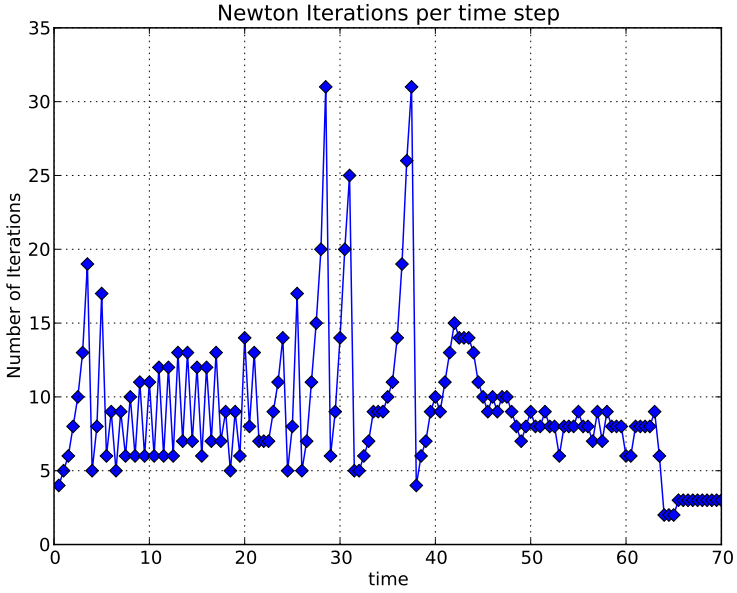

Figure 6.2: Number of Newton iterations with Jacobian reuse for the 2D Blood vessel problem.

6.2.2 Jacobian Simplification

One way to reduce the cost of assembling a Jacobian matrix is to simplify it by dropping terms. When doing this one must weigh the benefits of cheaper assemblies against the cost of additional Newton iterations due to reduced convergence, and the risk of non convergence. Good candidates for Jacobian simplification are usually expensive blocks with small entries that are located away from the main diagonal.

If we reexamine the structure of the Jacobian matrix in Figure 3.3, we can notice that the largest off diagonal terms are present in the blocks $\delta D_{_F}[v_{_F}]$ and $\delta D_{_F}[q_{_F}]$, representing the effect of the fluid domain equation on the fluid velocity and pressure. It was noticed in Fernández and Moubachir [2005] that the effect of these blocks is strongest at the FSI interface, where all of the physical movement of the fluid is represented by the movement of the fluid domain. Further away from the interface we can expect the smoothing effects of the fluid domain equation to increasingly dampen the size of $D_{_F}$, thereby decreasing the magnitude of the corresponding matrix entries. This observation inspired the removal of the non interface parts of the blocks $\delta D_{_F}[v_{_F}]$ and $\delta D_{_F}[q_{_F}]$.

In Table 6.4, runtime data is shown for the running of the test problems with the simplified Jacobian.

The data show a good gain in runtime for the small analytic problem

Table 6.4: Test problem runtimes with Jacobian simplification. Non interface entries of the blocks $\delta D_{_F}[v_{_F}]$ and $\delta D_{_F}[q_{_F}]$ have been removed from the Jacobian.

Problem	**Method**	**Calls**	**% runtime**
Analytic problem	Jacobian assembly:	50 (+22)	-40%
mesh vertices = 231	Linear solve:	50 (+22)	+89%
time steps = 10	Residual assembly:	60 (+22)	+49%
	Total Runtime:		**-44%**
Channel with flap	Jacobian assembly:	291 (+211)	-5%
mesh vertices = 6601	Linear solve:	291 (+211)	+318%
time steps = 40	Residual assembly:	331 (+211)	+172%
	Total Runtime:		**+8%**
2D Blood vessel			
mesh vertices = 1271	**Fails to Converge**		
time steps = 140			

when using the simplified Jacobian as opposed to the standard Newton's method. However, the data from the Channel with flap problem show that the cheaper Jacobian assemblies are not always worth the price. In this case the additional Newton iterations lead to a total 8 % increase in run time. For the Blood vessel problem using the simplified Jacobian lead to non-convergence already in the first time step. Due to the higher sensitivity in this system, the use of the full Jacobian is necessary.

On the basis of the gathered data the above presented Jacobian simplification scheme does not seem to be a very reliable way to improve run time. It does however aid with memory efficiency due to the increased sparsity of the Jacobian matrix. This gain might possibly be wasted by the fill-in of an LU solver, but will certainly be present if a Krylov method is used.

In the future, other FSI problems and or technological change might lead to this optimization being relevant. Also, there are many possible Jacobian simplification schemes, and it is very possible that one exists that is more efficient than the one above, at least in certain circumstances. An alternative simplification that achieved good results in some situations can be found in Degroote et al. [2009].

6.2.3 Jacobian Buffering

A small part of the Jacobian matrix is constant from time step to time step and can be buffered, that is saved and reused for every Newton iteration.

The variational forms that give constant contributions are

$$
\begin{aligned}
R'^{C}_{S} &= \left\langle c_{S},\ \ \rho_{S}\delta\dot{U}^{n}_{S} \right\rangle_{S} + \left\langle v_{S},\ \delta\dot{D}^{n}_{S} - \delta U^{n-\frac{1}{2}}_{S} \right\rangle_{S} \\
R'^{C}_{FD} &= \left\langle c_{F},\ \delta\dot{D}^{n}_{F} \right\rangle_{F} + \left\langle \mathrm{Grad}^{\mathbf{s}} c_{F},\ \ 2\mu_{FD}\mathrm{Grad}^{\mathbf{s}}\delta D^{n-\frac{1}{2}}_{F} + \lambda_{FD}\mathrm{tr}(\mathrm{Grad}\,\delta D^{n-\frac{1}{2}}_{F})I \right\rangle_{F} \\
R'^{C}_{\Gamma_{FSI}} &= \langle v_{F},\ \delta L_{U}\rangle_{\Gamma_{FSI}} + \langle m_{U},\ \delta U_{F} - \delta U_{S}\rangle_{\Gamma_{FSI}} \\
&\quad + \langle c_{F},\ \delta L_{D}\rangle_{\Gamma_{FSI}} + \langle m_{D},\ \delta D_{F} - \delta D_{S}\rangle_{\Gamma_{FSI}}
\end{aligned}
\tag{6.1}
$$

These terms come from the parts of the residual that are linear in terms of the unknown functions. The total constant Jacobian is given by $R'^{C} = R'^{C}_{S} + R'^{C}_{FD} + R'^{C}_{\Gamma_{FSI}}$.

As shown in Table 6.5, the Jacobian buffering scheme improves the total runtimes of the test problems by somewhere between 5% and 10%. This total increase is is due to faster Jacobian assembly times. Table 6.5 shows that the savings on Jacobian assembly time and total assembly time closely match on another, indicating that the buffering scheme works as it should. Theoretically the runtimes of the linear solve and residual assembly operations should remain unchanged, however some slight variation was observed. Note that the number of Newton iterations did not change with the use of the buffering scheme. This confirms that the convergence of the Newton's method is unaffected by Jacobian buffering.

As a price for the increased speed with Jacobian buffering we can expect an increased memory requirement due to the necessity of storing an additional matrix. The experiences made with this optimization indicate that it can be used in situations where memory is not an issue, without effecting the convergence properties of the Newton's method.

Table 6.5: Test problem runtimes with Jacobian buffering.

Problem	**Method**	**Calls**	**% runtime**
Analytic problem	Jacobian assembly:	28 (+0)	-7%
mesh vertices = 231	Linear solve:	28 (+0)	+3%
time steps = 10	Residual assembly:	38 (+0)	+1%
	Total Runtime:		**-6%**
Channel with flap	Jacobian assembly:	80 (+0)	-10%
mesh vertices = 6601	Linear solve:	80 (+0)	-5%
time steps = 40	Residual assembly:	120 (+0)	-5%
	Total Runtime:		**-10%**
2D Blood vessel	Jacobian assembly:	343 (+0)	-5%
mesh vertices = 1271	Linear solve:	343 (+0)	-1%
time steps = 140	Residual assembly:	438 (+0)	+0%
	Total Runtime:		**-4%**

6.2.4 Reduced order quadrature

If we examine a typical term from the variational forms associated to block $\delta D_{_F}[v_{_F}]$ of the Jacobian, for example

$$\rho_{_F} J_{_F} \mathrm{tr}(\mathrm{Grad}\ \delta U_{_F} \cdot F_{_F}^{-1})(\mathrm{Grad}\ U_{_F} \cdot F_{_F}^{-1} \cdot (U_{_F})) \tag{6.2}$$

we see that it consists of a product of many functions. The exact integration of such a term requires a high number of quadrature points which are in turn partially responsible for long Jacobian assemble times. If a high precision Jacobian is not required, than a cheaper Jacobian may be assembled by limiting the quadrature degree. The price for this is poorer convergence of the Jacobian, both in terms of a slower convergence rate and increased risk of blow up when using an imprecise Jacobian. In the numerical experiments conducted with the reduced order quadrature the order 2 was chosen as the limit of the quadrature degree. This limit was chosen as it was guessed that it would provide a significant reduction in Jacobian assembly time while still retaining good enough convergence properties. By default the quadrature order of a variational form that is assembled in DOLFIN is set just high enough so that the integration is exact.

Table 6.6 shows the results of running the test problems with the quadrature order reduced to 2. The results show a 74% and 63% reduction in runtime for the Analytic problem and Blood vessel. This speedup is completely due to the cheaper Jacobian assemblies which outweigh the cost of the extra iterations. It is interesting to note that the reduced quadrature scheme succeeds with the more challenging Blood vessel problem, but fails when applied to the easier Channel with flap.

Table 6.6: Test problem runtimes with quadrature order reduced to 2.

Problem	**Method**	**Calls**	**% runtime**
Analytic problem	Jacobian Assembly:	50 (+22)	-80%
mesh vertices = 231	Linear solve:	50 (+22)	+131%
time steps = 10	Residual assembly:	60 (+22)	+72%
	Total Runtime:		**-74%**
Channel with flap			
mesh vertices = 6601	**Fails to Converge**		
time steps = 40			
2D Blood vessel	Jacobian Assembly:	658 (+315)	-87%
mesh vertices = 1271	Linear solve:	658 (+315)	+40%
time steps = 140	Residual assembly:	798 (+315)	+60%
	Total Runtime:		**-63%**

ively

6.3 Summary of Optimizations

Figure 6.3 presents a summary of the optimizations that were considered in the previous sections. Thumbs up symbolizes an improvement over the standard Newton's method, thumbs down a worsening, and the sideways thumb indicates a neutral effect. Ratings in robustness reflect the relative danger of nonconvergence.

Optimization	*Runtime*	*Memory*	*Robustness*
Jacobian Reuse			
Jacobian Simplification			
Jacobian Buffering			
Reduced Quadrature Order			

Figure 6.3: Summary of the advantages and disadvantages of optimizations in FSINewton as compared to the standard Newton's method.

6.3.1 Recommended Use of Optimizations

Jacobian Reuse saves so much time it is recommended to always use it. The only (rare) exception to this would be the simulation of a system so highly chaotic that the reused Jacobians lead to too many blow ups, i.e. too many wasted iterations.

Jacobian Simplification does not seem very useful at the moment due to the danger of increasing the simulation time and the risk of non-convergence.

Jacobian Buffering can be used as long as there is sufficient memory.

Reduced Quadrature should be used carefully as it can drastically reduce the quality of the Jacobian.

Chapter 7

COMPARISON OF NEWTON'S METHOD AND FIXED POINT ITERATION

In this section the Newton's method algorithm in FSINewton is compared to the FSI fixed point algorithm in CBC.swing. It is shown that the two solvers have comparable runtimes, but that the Newton's method is the more robust method which succeeds in obtaining convergence when the fixed point solver does not. The results are compatible with previous numerical experiments, [Tallec and Mouro, 2001, Nobile, 2001, Deparis et al., 2003, Gerbeau and Vidrascu, 2003, Fernández and Moubachir, 2005], which have shown that only fully coupled schemes which implicitly deal with the change in fluid geometry and interface conditions ensure the necessary stability needed to efficiently solve FSI problems where the fluid and structure densities are of the same order.

7.1 Brief Description of the Fixed Point Iteration Solver

The fixed point solver in CBC.swing uses a partitioned approach to solving FSI problems, which is illustrated in figure 7.1. In this framework there are three separate solvers, one for the fluid subproblem, one for the structure subproblem and one for the mesh subproblem. The mesh subproblem corresponds to our fluid domain equation 2.34; the difference being that the fixed point solver moves the fluid mesh at each time step so that the fluid equation is solved in the current domain $\omega_{_F}(t)$, rather than in the reference domain $\Omega_{_F}$.

The fluid subproblem is solved either using Taylor Hood elements and Newton's method, similarly to the FSINewton, or using an operator splitting method called the incremental pressure correction scheme (icps) [Katuhiko, 1979]. The structure subproblem is solved using 1st order elements and Newton's method. Finally the mesh subproblem, being linear, is solved for directly, using 1st order elements.

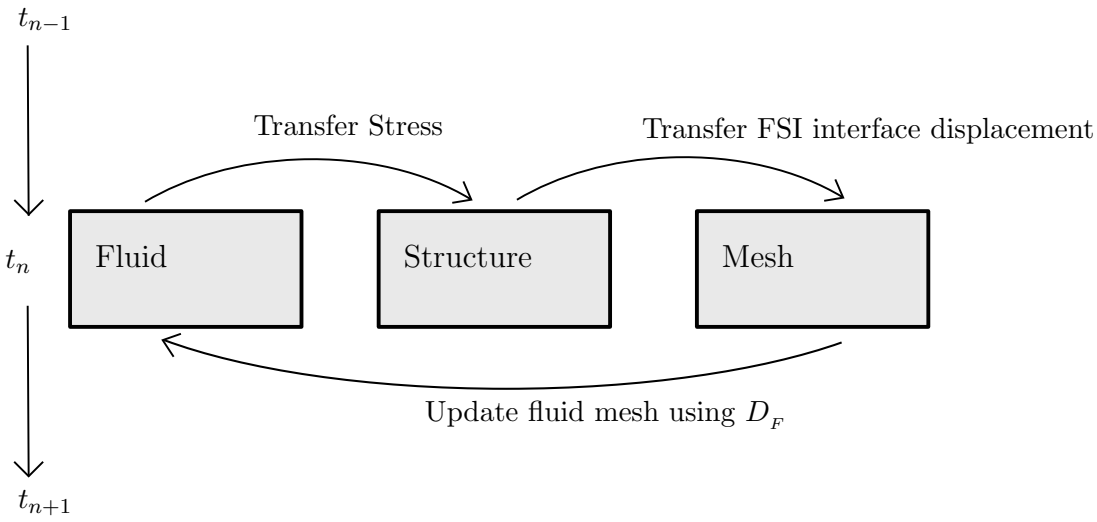

Figure 7.1: The partitioned algorithm of the CBC fixed point solver. In each time step k_n the three subproblems are solved iteratively using a simple fixed point method.

Mathematically the fixed point solver is based upon the contraction mapping principle. This principle states that if an operator A satisfies the contraction mapping property

$$\|Ax - Ay\| \leq C\|x - y\| \tag{7.1}$$

where x and y are arbitrary points in a normed space, and $C < 1$, then there exists a unique point x^* such that $Ax^* = x^*$. The operator A here correspond to one iteration of the fixed point cycle, and the fixed point x^* corresponds to the system state at the time level t_n. A proof of the contraction mapping theorem can be found in Hunter and Naechtergale [2001]. More details about the fixed point algorithm and implementation can be found in Selim [2012].

7.2 Runtime Comparison

Table 7.1 displays the runtimes of the three test problems using the fixed point and Newton's method solvers. The Newton's method solver uses only

Table 7.1: Test problem runtimes, optimized Newton vs. Fixed point solver

Solver	**Analytic**	**Channel**	**2D Vessel**
Newton	6.6788 s	307.7338 s	1036.948 s
Fixed point	8.8281 s	477.3729 s	690.54 s

the Jacobian reuse optimization. The fixed point solver uses the relatively fast icps fluid subsolver for the first two test problems. For the last test problem, the 2D blood vessel, the relatively slow Taylor-Hood subsolver is used since the icps solver fails to converge.

The runtimes are quite comparable for the Newton's method and fixed point solvers. The easier analytic and channel with flap problems are run faster by the Newton's solver, which only needs to execute a single Jacobian assembly for each problem. However, for the more challenging 2D vessel problem the Newton solver, needing to reassemble 25 Jacobians, is actually slower than the fixed point solver.

7.3 Robustness Comparison

In numerical analysis one often characterizes methods as *implicit* or *explicit*. Implicit methods are those that at each time level solve a system of equations involving the current and past system states. Explicit methods on the other hand just use past system state information to calculate the current state. Typically implicit methods are more computationally expensive than explicit methods, but are more robust, allowing for the usage of larger time steps for *stiff* systems. A system is said to be stiff if it exhibits a high degree of

numerical instability, or sensitivity. This can force an implicit method to use such small time steps that an explicit method is preferable. In the context of our FSI solvers the Newton's method is implicit, whereas the fixed point method is explicit.

In order to confirm the superior robustness of the Newton's method a stress test was designed using the 2D blood vessel problem. In this test the numerical instability in the system was increased by lowering the structure-fluid density ration from 4 to 2. The time step size was then progressively increased, and the number of iterations required to solve the 1st time step were plotted, (see Figure 7.2). At around time levels 3.0 and 3.5 we can see the iteration count exploding for the fixed point solver, where as the iteration count of the full Newton's method is unaffected, and the iteration count of the reuse Newton's method is only slightly worsened.

In Figure 7.3 we compare the robustness of the simplified Newton's method described in 6.2.2, to the fixed point method. The results show that the simplification of the Jacobian has drastic effects on the robustness of the Newton's method. In order to obtain convergence it was necessary to increase the density ration to 4. Even then the simplified Jacobian method fails to converge with a modest time step of 0.4.

Studies similar to the one above have been documented in other articles, namely Fernández and Moubachir [2005] and Heil [2004], both of which confirm the superior robustness of the full Newton's method.

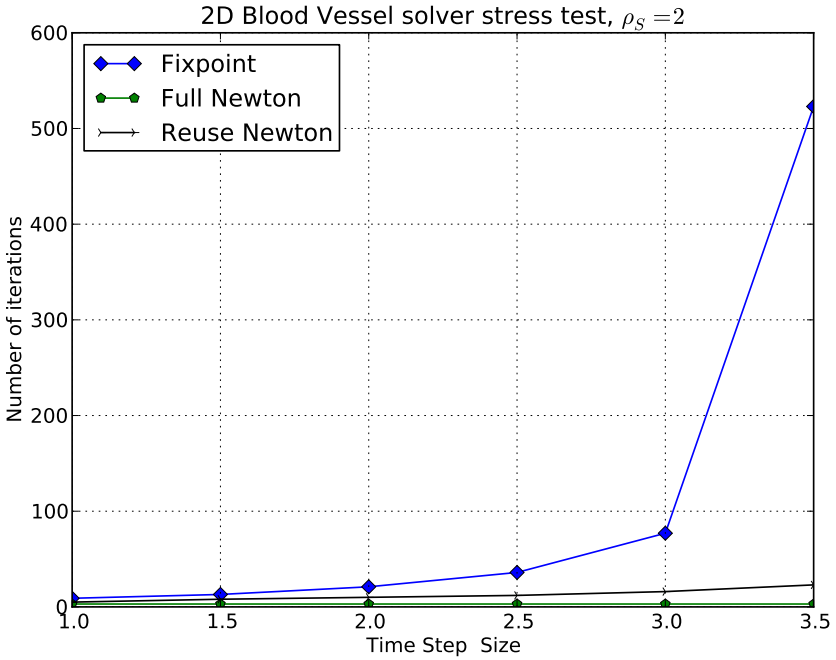

Figure 7.2: Robustness testing of the fixed point solver vs. Newton solvers. Increased time step size causes fixed point solver failure.

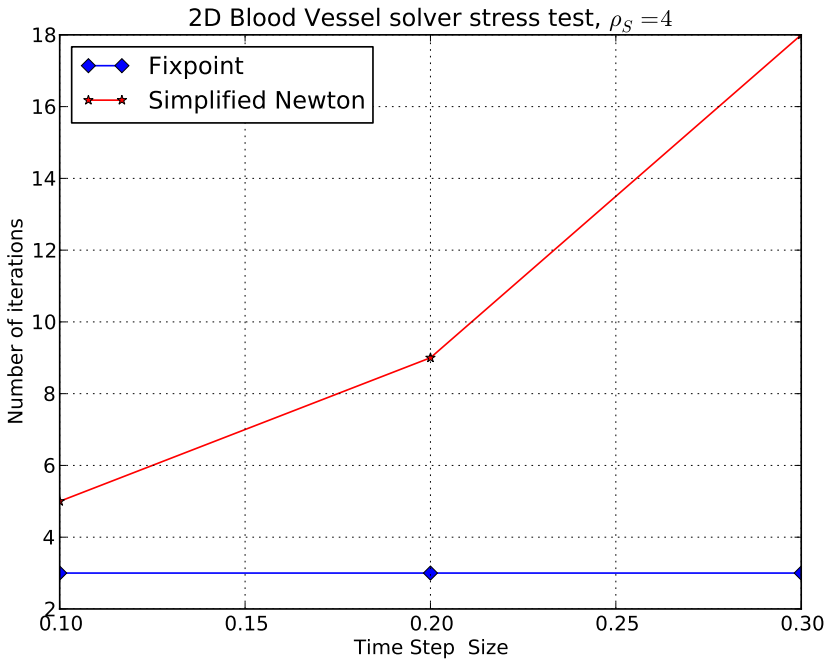

Figure 7.3: Robustness testing of the fixed point solver vs. the simplified Jacobian Newton's method. Increase time step size causes the simplified Jacobian Newton's method to fail first.

Chapter 8

CONCLUSIONS AND FUTURE WORK

In this thesis a finite element Newton's method based solution algorithm was created for fluid-structure interaction using an ALE formulation. This algorithm was implemented as a software package, FSINewton, and made available online via the open source biomedical solver framework, CBC.Solve.

Three test problems were formulated and solved to test the computer implementation of Newton's method for FSI; the Analytical, Channel with Flap, and 2D Blood vessel problems. The last two problems were used to show that Newton's method can solve both easy and difficult FSI problems, while the Analytical problem was used to confirm the correctness of the implementation by obtaining L^2 convergence of the finite element solutions to the known analytic solution.

Furthermore, four run-time optimizations to FSINewton were implemented, namely Jacobian reuse, Jacobian simplification, Jacobian buffering, and reduced order quadrature. These optimizations were tested using the three test problems. On the basis of the observations made the four optimizations were evaluated, with the Jacobian reuse scheme coming out as the most valuable optimization.

Finally, the FSI fixed point solver of CBC.Solve was compared to the Newton solver of this thesis. Comparable run-times were obtained for both solvers, and an example was given to demonstrate the superior robustness of the Newton's method solver in the case of fluids and structures with similar densities.

In the near future an extension of the FSINewton framework to 3-D problems is planned. Several other projects however, remain open for the future. Chief among these is the creation of an effective block preconditioning scheme to allow the use of Krylov methods in FSINewton. Doing this would most likely greatly save on memory, thereby allowing the simulation of larger systems. Further enhancements to the code could be made by incorporating parallel processing techniques in order to further speed up the run time. Also, distributed computing techniques could be used to recruit the memories of several computers in order to store an even larger Jacobian

matrix.

Other possible expansions of the FSINewton framework are a fluid model for potential flow, which greatly reduces the number of fluid degrees of freedom in the case that the fluid velocity field is irrotational. For problems involving a small structure that does not move much, such as the channel with flap problem, it is possible to use a mixed Eulerian/ALE framework to describe the fluid flow. This involves modeling fluid regions far away from the interface with a pure Eulerian description, and modeling regions close to the fluid-structure interface with the ALE description. This works as long as no fluid domain deformation is required in the Eulerian regions, and saves on the degrees of freedom associated to the fluid domain mapping. Finally it might be worthwhile experimenting with the MINI finite element [Arnold et al., 1984] for the discretization of the fluid equations, as the MINI element contains fewer degrees of freedom than the Taylor-Hood element does.

At the moment the Jacobian reuse scheme is steered by a user controlled maximum limit on the number of Jacobian iterations before reassembly. The selection of this parameter could be automated by having the computer keep track of the run times of Jacobian assemblies, residual assemblies, and linear solves. This data could be used to then find the optimal point in a simple model that predicts the total run time based on when the Jacobian is reassembled.

Finally, further work is also necessary on the enforcement of the two Dirichlet type interface conditions. It would be desirable to have some way of controlling how strictly the conditions are imposed. A possible alternative to the Lagrange multiplier formulation in this case would be an interior penalty method, which could vary the strictness of the interface conditions via a user controlled penalty parameter.

Bibliography

On newton's method for functional equations. *Dokl. Akad. Nauk*, 59:1237 – 1240, 1948.

Martin S. Alnæs. *UFL: a Finite Element Form Language*, chapter 17. Springer, 2012.

Martin S. Alnaes, Anders Logg, Kent-Andre Mardal, Ola Skavhaug, and Hans Petter Langtangen. Unified framework for finite element assembly. *International Journal of Computational Science and Engineering*, 4(4): 231–244, 2009. doi: 10.1504/IJCSE.2009.029160.

Martin S. Alnæs, Anders Logg, and Kent-Andre Mardal. *UFC: a Finite Element Code Generation Interface*, chapter 16. Springer, 2012.

Martin Sandve Alnæs. *A Compiler Framework for Automatic Linearization and Efficient Discretization of Nonlinear Partial Differential Equations*. PhD thesis, University of Oslo, Unipub, Oslo, Norway, September 2009. ISSN 1501-7710, No. 884.

D.N. Arnold, F.Brezzi, and M.Fortin. A stable finite element for the stokes equations. *Estratto da Calcolo*, 21(4):127–142, 1984.

T.B. Belytschko and J.M. Kennedy. Computer models for subassembly simulation. *Nuclear Engineering and Design*, 49(1–2):17 – 38, 1978. ISSN 0029-5493. doi: 10.1016/0029-5493(78)90049-3.

Hans-Joachim Bungartz and Michael Schaefer, editors. *Fluid-Structure Interaction Modelling, Simulation, Optimization*, march 2006.

Joris Degroote, Klaus-Juergen Bathe, and Jan Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. *Computers &; Structures*, 87(11–12):793 – 801, 2009. ISSN 0045-7949. doi: 10.1016/j.compstruc.2008.11.013. Fifth MIT Conference on Computational Fluid and Solid Mechanics.

Simone Deparis, Miguel Á. Fernández, and Luca Formaggia. Acceleration of a fixed point algorithm for fluid-structure interaction using transpiration

conditions. *Mathematical Modelling and Numerical Analysis*, 37(4):601–16, 2003. doi: 10.1051/m2an:2003050.

W. Dettmer and D. Perić. A computational framework for fluid–rigid body interaction: Finite element formulation and applications. *Computer methods in applied mechanics and engineering*, 195:1633–1666, 2006.

Wulf G. Dettmer and Djordje Perić. On the coupling between fluid flow and mesh motion in the modelling of fluid–structure interaction. *Journal of Computational Mechanics*, 30(1):81–90, 2008.

J. Donea, S. Giuliani, and J.P. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33(1–3):689 – 723, 1982. ISSN 0045-7825. doi: 10.1016/0045-7825(82)90128-1".

Thomas Dunne and Rolf Rannacher. Adaptive finite element approximation of fluid-structure interaction based on an eulerian variational formulation. In Bungartz and Schaefer [2006], pages 110 – 146.

Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 2nd edition, 2010.

M.Á. Fernández and M. Moubachir. A Newton method using exact jacobians for solving fluid–structure coupling. *Computers & Structures*, 83(2):127–142, 2005.

Baijens F.P.T. A fictitious domain/mortar element method for fluid-structure interaction. *International Journal of Numerical Methods in Fluids*, 35(7):743–761, 2004.

J.F. Gerbeau and M. Vidrascu. A quasi newton method based on a reduced model for fluid structure problems in blood flows. *Technical Report INRIA*, 4691, 2003.

Morton E. Gurtin. *An introduction to continuum mechanics*. Academic Press, New York, 1981. ISBN 0123097509.

Matthias Heil. An efficient solver for the fully-coupled solution of large-displacement fluid-structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 193(1–2):1 – 23, 2004. ISSN 0045-7825. doi: 10.1016/j.cma.2003.09.006.

C.W Hirt, A.A Amsden, and J.L Cook. An arbitrary lagrangian-eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227 – 253, 1974. ISSN 0021-9991. doi: 10.1016/0021-9991(74)90051-5.

Jaroslav Hron and Stefan Turek. A monolithic fem/multigrid solver for an ale formulation of fluid-structure interaction with applications to biomechanics. In Bungartz and Schaefer [2006], pages 146 –171.

John K Hunter and Bruno Naechtergale. *Applied Analysis*. World Scientific Publishing Company, 2001. ISBN 9810241917, 978-9810241919.

I.Babuška. The finite element method with lagrangian multipliers. *Numerical Mathematics*, 20:179–192, 1973.

J.-Ph. Ponthot J. Donea1, Antonio Huerta and A. Rodríguez-Ferran. *Encyclopedia of Computational Mechanics*. Wiley, 2004. ISBN 9780470091357.

Goda Katuhiko. A multistep technique with implicit difference schemes for calculating two-or three-dimensional cavity flows. *Journal of Computational Physics*, 30(1):76–95, 1979. ISSN 0021-9991.

Robert C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, 2004. doi: 10.1145/1039813.1039820.

Robert C. Kirby. *FIAT: Numerical Construction of Finite Element Basis Functions,*, chapter 13. Springer, 2012.

Robert C. Kirby and Anders Logg. A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3), 2006. doi: 10.1145/1163641.1163644.

A. Legay, J. Chessa, and T. Belytschko. An eulerian–lagrangian method for fluid–structure interaction based on level sets. *Computer Methods in Applied Mechanics and Engineering*, 195(17–18):2070 – 2087, 2006. ISSN 0045-7825. doi: 10.1016/j.cma.2005.02.025.

Jie Li, Marc Hesse, Johanna Ziegler, and Andrew W. Woods. An arbitrary lagrangian eulerian method for moving-boundary problems and its application to jumping over water. *Journal of Computational Physics*, 208(1): 289 – 314, 2005.

Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012a. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8.

Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. *FFC: the FEniCS Form Compiler*, chapter 11. Springer, 2012b.

A. Massing, M.G.Larson, A.Logg, and M.E.Rognes. A stabilized nitsche overlapping mesh method for the stokes problem. 2012a.

A. Massing, M.G.Larson, A.Logg, and M.E.Rognes. A stabilized nitsche fictitious domain method for the stokes problem. 2012b.

Hermann G. Matthies and Jan Steindorf. Partitioned but strongly coupled iteration schemes for nonlinear fluid–structure interaction. *Computers &; Structures*, 80(27–30):1991 – 1999, 2002. ISSN 0045-7949. doi: 10.1016/S0045-7949(02)00259-6.

Hermann G. Matthies and Jan Steindorf. Partitioned strong coupling algorithms for fluid–structure interaction. *Computers &; Structures*, 81(8–11): 805 – 812, 2003. ISSN 0045-7949. doi: 10.1016/S0045-7949(02)00409-1. K.J Bathe 60th Anniversary Issue.

Hallgeir Melbø and Trond Kvamsdal. Goal oriented error estimators for stokes equations based on variationally consistent postprocessing. *Computer Methods in Applied Mechanics and Engineering*, 192(5–6):613 – 633, 2003. ISSN 0045-7825. doi: 10.1016/S0045-7825(02)00575-3.

M. Nazem, J.P. Carter, and D.W. Airey. Arbitrary lagrangian–eulerian method for dynamic analysis of geotechnical problems. *Computers and Geotechnics*, 36(4):549 – 557, 2009. ISSN 0266-352X. doi: 10.1016/j.compgeo.2008.11.001.

Fabio Nobile. *Numerical approximation of fluid-structure interaction problems with application to haemodynamics*. PhD thesis, Lausanne, 2001. URL `http://library.epfl.ch/theses/?nr=2458`.

W.F. Noh, B. Alder, S. Fernbach, and M. Rotenberg (Eds.). A time-dependent two-space-dimensional coupled eulerian–lagrangian code. *Methods in computational physics*, 3, 1964.

Kristian B. Ølgaard and Garth N. Wells. Optimisations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software*, 37, 2010. doi: 10.1145/1644001.1644009.

Charles S. Peskin. The numerical analysis of bloodflow in the heart. *Journal of Computational Physics*, 25(3):220–252, 1977.

Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 1(1): 479–517, 2002.

Rolf Rannacher. Finite element methods for the incompressible navier-stokes equations. Lecture Notes `numerik.uni-hd.de/Oberwolfach-Seminar/CFD-Course.pdf`, 1999.

R.Glowinski, T.W. Pan, and J. Periaux. A distributed lagrange multiplier/-fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 25(5):755–794, 1999.

S.Deparis. *Asymmetric flow in moving domains and fluid-structure interaction algorithms for blood flow modeling.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2004.

S.Deparis, J.F. Gerbeau, and X. Vasseur. Dynamic gmres based preconditioner for newton or quasi newton algorithms with application to fluid structure interaction. *Technical Report INRIA*, 5352, 2004.

Kristoffer Selim. *Adaptive Finite Element Methods for Fluid–Structure Interaction and Incompressible Flow.* PhD thesis, University of Oslo, 2011.

Kristoffer Selim. An adaptive finite element solver for fluid–structure interaction problems. In Anders Logg, Kent-Andre Mardal, Garth Wells, Timothy J. Barth, Michael Griebel, David E. Keyes, Risto M. Nieminen, Dirk Roose, and Tamar Schlick, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, pages 553–569. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-23099-8.

J.L Stegera, F.C. Dougherty, and J.A. Benek. A chimera grid scheme. *Advances in Grid Generation*, ASME FED-5.

Leland R. Stein, Richard A. Gentry, and Cyril W. Hirt. Computational simulation of transient blast loading on three-dimensional structures. *Computer Methods in Applied Mechanics and Engineering*, 11(1):57 – 74, 1977. ISSN 0045-7825. doi: 10.1016/0045-7825(77)90068-8.

P. Le Tallec and J. Mouro. Fluid structure interaction with large structural displacements. *Computer Methods in Applied Mechanics and Engineering*, 190(24–25):3039 – 3067, 2001. ISSN 0045-7825. doi: 10.1016/S0045-7825(00)00381-9.

C. Taylor and P. Hood. A numerical solution of the navier-stokes equations using the finite element technique. *Computers &; Fluids*, 1(1):73 – 100, 1973. ISSN 0045-7930. doi: 10.1016/0045-7930(73)90027-3.

Tayfun E. Tezduyar. Finite element methods for fluid dynamics problems with moving boundaries and interfaces. *Archives of Computer Methods in Engineering*, 8(2):83 – 120, 2001.

Wolfgang A. Wall, Axel Gerstenberger, Peter Gamnitzer, Christiane Foerster, and Ekkehard Ramm. Large deformation fluid-structure interaction-advances in ale methods and new fixed grid approaches. In Bungartz and Schaefer [2006], pages 195 – 233.

Frank M. White. *Viscous Fluid Flow.* McGraw-Hill Companies,Incorporated, 2005. ISBN 0072402318, 9780072402315.